

# Uses of Randomness in Algorithms and Protocols

by

Joe Kilian

B.S. Mathematics and Computer Science  
Massachusetts Institute of Technology  
(1985)

Submitted to the Department of Mathematics  
in Partial Fulfillment of  
the Requirements for the Degree of  
Doctor of Philosophy in Mathematics  
at the  
Massachusetts Institute of Technology  
April 1989

Massachusetts Institute of Technology 1989  
all rights reserved

Signature of Author \_\_\_\_\_  
Department of Mathematics  
April 3, 1989

Certified by \_\_\_\_\_  
Shafi Goldwasser  
Associate Professor of Computer Science

Certified by \_\_\_\_\_  
David Shmoys  
Assistant Professor of Mathematics

Accepted by \_\_\_\_\_  
Daniel J. Kleitman  
Chairman, Applied Mathematics Committee

Accepted by \_\_\_\_\_  
Sigurdur Helgason  
Chairman, Departmental Graduate Committee

# Uses of Randomness in Algorithms and Protocols

by

Joe Kilian

Submitted to the Department of Mathematics  
on April 3, 1989 in partial fulfillment of the requirements for the degree of  
Doctor of Philosophy in Mathematics

## Abstract

We develop techniques for using randomness in algorithms and the design of secure protocols. First, we exhibit a probabilistic algorithm for generating large certified primes. Next, we give a round efficient reduction from secure circuit evaluation to oblivious transfer. Finally, we study the properties of a multi-prover generalization of interactive proof systems.

Thesis Supervisor: Dr. Shafi Goldwasser

Title: Associate Professor of Computer Science

## Acknowledgements

It is customary to, after completing ones dissertation, acknowledge ones friends, family, companions, and other benefactors for all of their help and support. I have frighteningly many people to be grateful to, and this thesis is too long as it is. I give then a very meager listing, and extend my general thanks to the many people who helped me make it to this point in my life.

First, my parents, Mary and Leonard Kilian. Mom and Dad. You, of course, started me out, which was easy, and I hope enjoyable. More importantly, you cared for me, guided me, and loved me for the entire duration. You're both slightly crazy, but I love you very, very much anyway.

My brothers: Mike, John, Danny and Steven. You guys are the reason Mom and Dad are slightly crazy. I am not at all responsible. Still, my life would not have been the same without you, and just this once, I'd like to thank you all. Don't let it go to your heads.

Mr. Frank Kess. I wish you could have seen this. Thanks for your questions. "What is the volume of a pyramid?" "How can you compute the great circle distance between two points on the earth?" You were a man to be looked up to, and for those of us who remember, you still are.

The Hampshire College Summer Studies in Mathematics program, which helped my development in mathematics in at least 17 ways.

The residents of Random Hall, my surrogate family in Cambridge. I will never again fear living between a gas station and a bar. I could easily fill pages and pages with names and dates and 4+ years worth of wild stories. Instead, I'll wait till one of you runs for elected office, and in the meantime thank you all.

Professor Mike Sipser. It's been seven years. Kind of frightening. Thanks for turning me on to finite automata, and for all your help and friendship throughout the years. I only regret never having tried out your sauna.

Professor Albert Meyer. As far as I know, there is no purple heart awarded to undergraduate thesis advisors, but if the rules change I'll write a letter on your behalf. Thanks for never putting me in a straightjacket, or forcibly injecting me with Thorazine. I'm sure I gave you just cause more than a few times. In lieu of a medal, please accept my belated thanks. I have no doubt that Shafi had it slightly easier because of you.

Arline Benford, OGB. What more is there to say? Thanks for all the hugs, kisses, and for being such an incredibly cutey while making me feel

like one' too. You are the British Empire's finest addition in well over half a century.

Sally Bemus. Thanks for all the dinners at Bertucci's and the Royal East, and for being sympathetic and nice to me even when the computers weren't.

Be Hubbard. AKA Mom II. You are such a sweetheart. Our work on using bilateral symmetry to achieve optimal hand/time tradeoffs remains one of my favorite pieces of research. People will probably wonder what kinky acts I'm referring to. Let them.

The faculty and students of the MIT Theory of Computation group. There are just a few basic things that make a great research environment. The three biggies are comfortable couches, free hot chocolate, and wonderful friends and coworkers. To name just a few: Charles Leiserson, Silvio Micali, Ray Hirschfeld, Claude Crépeau, Tom Leighton, Bonnie Berger, Miller Maley, Su Ming Wu, Ron Rivest, Aditi Dhagat(AKA Dina), Dina Kravets(AKA Aditi), Bonnie Eisenberg, RObert Ashcroft, Bruce Maggs, Alex Ishii, Tom Cormen, Bard Bloom, Lenore Cowen, and zillions and zillions of others. You guys kept me sane. I only wish I could have returned the favor.

Finally, professor Shafi Goldwasser. You have been an excellent collaborator, advisor, and friend. Without you, this thesis would have never been possible.

# Chapter 1

## Introduction to the thesis.

In this chapter, I give a brief introduction to the main body of this thesis. In Section 1.1, I introduce a common theme to the three pieces of work that comprise this thesis. In Section 1.2, I introduce the first topic, primality testing, and very briefly discuss its historical context. In Section 1.3 I give some of the background, both historical and informational, necessary for the understanding the cryptography portion of the thesis. In Section 1.4 I introduce the second topic of this thesis, secure two-party computation, and gives its historical context. In Section 1.5 I briefly discuss the third topic of this thesis, multi-prover interactive proof systems. The reader may note some repetition in material between this introduction, and the chapter material. The intent here is not to strive for added length, but to make the introduction more self-contained.

### 1.1 How many theses have I written?

This thesis covers a wide span of material, ranging from algorithms to secure protocols to zero-knowledge proof systems. Such a mishmash seems to preclude any unifying theme, and one might naturally suspect that this thesis is, in reality, three theses bundled together. One might, with no meanness of spirit, charge that the sole unifying force behind these works is a staple. One might be right.

But then, one might be wrong. In putting these disparate papers side by side, one sees that there is in fact a common general theme. All three

areas of work are, in their own way, giving answers to the question, “*What can randomness do for me?*”. In the first area, randomness is applied to solve and illuminate some longstanding questions in the field of algebraic algorithms. In the second, I examine a simple, randomized mechanism for destroying information, and show how to use it to implement certain types of secure protocols. Finally, in the third area, I delve into the curious connections between information and belief afforded by randomized generalizations of the usual notions of proof.

Also common to these three works is a propensity for using complicated arguments to illuminate the power of very simple mechanisms. In the first area, this mechanism is a random bit generator, or, more colloquially, a common coin. In the second, this mechanism is a communication channel that transmits each bit sent through it with probability  $\frac{1}{2}$ . In the third area, I consider the more abstract mechanism of separation, examining the power obtained by blocking all communication between two parties.

### **An apology made, then rescinded.**

I sometimes feel I should apologize for the tedious intricacies of some of the work described in this thesis. A quick perusal through even the most monstrous chapter<sup>1</sup> reveals that nothing particularly ingenious is going on. Rather, there is page after page of grungy argumentation, whose sole purpose is to keep the thesis above some minimal level of rigor.

However, this complexity is often the price one must pay if one is to create new tools. The reduction from complicated algorithms and protocols to simple mechanisms or methodologies is often painful, but is productive in the long run. Particularly in the cryptography sections, I hope that by proving the strongest theorems I can, in a reasonably rigorous fashion, I may facilitate any subsequent work in these areas.

---

<sup>1</sup>Chapter 3, for the bold.

## **1.2 Part One: Primality Testing.**

### **1.2.1 Introduction.**

Chapter 2 of this thesis relates some work of Goldwasser and myself [GK] on the problem of distinguishing prime numbers from composites. Since this is a fairly well-known problem in the area of algebraic algorithms, and the chapter's introduction is already fairly long, this introduction will be brief. In this section, we will primarily discuss the changes in mathematical technology, and the shift in focus that made our results possible.

### **Prerequisite background for reading this chapter.**

Since our algorithm makes use of some deep results from the theory of elliptic curves, a strong background in algebraic geometry is recommended for reading this chapter of the thesis. Of course, a strong background in algebraic geometry was recommended for writing this chapter, but turned out not to be necessary. Thus someone with some undergraduate background in algebra should not lose hope.

I have attempted to suppress as much as possible the advanced mathematics behind this chapter, and to concentrate it into as small a space as possible. Indeed, the exposition of our basic methodology uses only elementary group theory, making no mention at all of elliptic curves. When I discuss the application of our methodology to the study of elliptic curves, I try to use their properties as abstract “black boxes.” In Section 2.3, I give a quick introduction to the mathematics behind elliptic curves. This section gives only a taste of a very rich field, and frequently references classical theorems without giving any proofs or even an idea for why they should be true. However, this section will prove sufficient for an understanding of our algorithm, if not the mathematics behind it.

### **1.2.2 Historical context.**

Historically, the work of Professor Goldwasser and myself, in the latter half of 1985, was part of a trend in developing algorithms using results from algebraic geometry. To give some context, we mention other work that also followed this trend.

Schoof [Sch] used the theory of elliptic curves to develop the first deterministic polynomial time algorithm for computing the square root of  $x \in \mathbb{Z}_p$ , where  $x$  is a small quadratic residue mod  $p$ .<sup>2</sup> This algorithm is less efficient than the probabilistic algorithms for this problem (e.g., [AMM]), and much more limited in the values of  $x$  that it is fast on. However, the previous approaches to this problem all require randomization in lieu of some unproven assumption, such as the Extended Riemann Hypothesis. More importantly, he developed an algorithm for computing the order of an elliptic curve, an algorithm which proves crucial to our work.

Lenstra [L] showed how to use elliptic curves to efficiently factor numbers using almost constant memory. Unlike most of the faster algorithms, Lenstra's technique works particularly well on numbers which have small prime factors, hence its practical importance. This work was particularly important to the development of our work; theorems he proved for the analysis of his algorithm are instrumental for the analysis of ours.

Finally, some earlier work by Bosma [Bo], and Chudnovsky-Chudnovsky [CC], while not directly influencing our work, also applied elliptic curves to primality testing.

### 1.2.3 A shift in focus.

While our work would not have been possible without the recent advances in elliptic curve techniques, we equally benefitted from a shift in focus. Previous primality testing algorithms ([M],[R],[SS]) worked by efficiently recognizing composite numbers. Given a number, they would attempt to find a short proof that the number was composite. If they succeeded, they would output "composite." If they failed to find a proof of primality, they would output "probably prime."

Such an approach is eminently reasonable if one can manage to *always* find a proof of compositeness for a composite number,  $n$ . In such a case, failing to find a proof of compositeness for  $n$  is in itself a proof that  $n$  is prime. However, all the provably fast strategies for proving a number composite use randomization,<sup>3</sup> which causes them to fail with nonzero probability. In prac-

---

<sup>2</sup>To better understand the meaning of "small" in this context, we note that the running time of the algorithm is polynomial in  $x$  and  $|p|$ , as opposed to  $|x|$  and  $|p|$ .

<sup>3</sup>An algorithm of Miller [M] uses a deterministic strategy, but its analysis relies on an unproven assumption.

tice, the difference between always succeeding and almost always succeeding is not crucial. However, this nonzero chance of failure means that failing to prove  $n$  composite does not imply that  $n$  is prime.

We used a different strategy altogether. Instead of attempting to prove a number composite in a more efficient manner, we worked on directly proving a number prime. We developed a general methodology for proving a number prime, which can be used to formulate a primality criterion based on elliptic curves. The development, use, and analysis of this criterion, is worked out in the main body of the thesis.

## 1.3 Introduction to modern cryptography.

The other two areas discussed in this thesis, comprising the bulk of this thesis, are in the area of cryptography. Papers in modern cryptography frequently have a few simple conceptual ideas, with long arguments to show that these ideas do indeed work.<sup>4</sup> For better, or for worse, this paper is no exception.

To facilitate the reading of these sections, we first give a general introduction to the field, briefly discussing the dominant issues we address, and the general proof techniques we use.

### 1.3.1 Trends in modern cryptography.

Cryptography has gone through many changes in its history, which is understandable given that the field is at least two or three thousand years old. Even in the “modern” era of the cryptography, which began in the seventies,<sup>5</sup> there has been a substantial change in focus. We have always been interested in privacy, but the types of privacy we consider are now much more sophisticated.

The traditional goal of cryptography has been to achieve private communication in public settings. In the standard scenario, we have two parties, Alice and Bob, who wish to have a private conversation, and an eavesdropper, Eve, who wishes to listen in on the conversation. We assume that Eve is allowed to listen to every message sent between Alice and Bob, but is not

---

<sup>4</sup>Actually, a cryptography paper is doing well if it manages to fit in all the proofs.

<sup>5</sup>The beginning of the modern era is typically tied to seminal paper, *New Directions in Cryptography*, by Diffie and Hellman [DH].

allowed to perform arbitrarily difficult computations. The bread and butter problem cryptographers work on is to find ways in which Alice and Bob can scramble their messages to prevent Eve from learning anything about what they are saying. Interestingly, this goal was not fully reached until the middle 70's, with the public key cryptosystems of [RSA] and [Mc].

Nothing presented in this thesis is directly applicable to private communication. Instead, we focus on issues that first reached prominence in the 80's: *zero-knowledge proofs* and *secure protocols*. Zero-knowledge proofs attempt to realize the distinction between the transfer of confidence and the transfer of knowledge. In secure protocols, each party possesses some information that they wish to keep private. The parties interact so as to compute some function of their private inputs, while keeping this data as private as possible. We discuss these concepts in more detail below.

### 1.3.2 Zero-knowledge proofs.

Chapters 3 and 6 of this thesis devote themselves almost entirely to the subject of zero-knowledge proofs. We should therefore give a rough idea of just what we mean by zero-knowledge proofs.<sup>6</sup> This concept was introduced by Goldwasser, Micali, and Rackoff [GMR], and in fact comprises two distinct conceptual advances.

#### Interactive proof systems.

First, there is the notion of an *interactive proof system*, a randomized generalization of the classical notion of proof. We consider assertions of the form “String  $x$  is in language  $L$ .” Interactive proofs generalize ordinary proofs in the following three ways:

1. In ordinary proof systems, the prover sends a single message to the verifier, who then deterministically evaluates this message without any further interaction with the prover. In an interactive proof system, the prover and the verifier may send messages back and forth to each other. Both the prover and the verifier are allowed to base their messages on

---

<sup>6</sup>The introduction and definitions section of Chapter 6 gives a more extensive and technical treatment of this subject. We point the more sophisticated reader in this direction.

the outcome of random coin tosses, i.e. both entities are probabilistic. The verifier makes his evaluation based on the transcript of the protocol, and his set of random coin tosses.

2. Suppose that  $x \in L$ . With an ordinary proof system, the verifier is guaranteed to accept a correct proof with probability 1. With an interactive proof system, we require only that the verifier accepts with probability close to 1 at the end of the protocol. We are being admittedly vague about how close to 1 this probability should be, for reasons that will be made clearer later.
3. Suppose that  $x \notin L$ . With an ordinary proof system, the verifier is guaranteed to reject a false proof with probability 1. With an interactive proof system, we require only that the verifier rejects with probability close to 1 at the end of the protocol, regardless of the malicious prover's strategy.

Now to clean up some details. By “close to 1,” we turn out to have a lot of leeway. Whether “close to 1” means greater than .99, or greater than .50001, the set of assertions that may be interactively proven usually remains the same.<sup>7</sup> We also require that the number of messages and the size of the messages remains polynomial in the size of the assertion to be proven. Thus, interactive proof systems are better thought of as generalizations of *NP*. Finally, when we refer to arbitrary strategies on the part of the malicious provers, we allow the prover to compute arbitrary functions and realize arbitrary distributions,<sup>8</sup> but we don't allow the prover access to the random coin tosses of the verifier.<sup>9</sup>

### Zero-knowledge proof systems.

Second, there is the notion of *zero-knowledge*. Informally, a zero-knowledge proof system is one in which the verifier receives no information other than

---

<sup>7</sup>This is true in nearly all the models considered in the literature, as well as in this thesis.

<sup>8</sup>Alternative models, such as the Brassard-Cr peau model [BC], only allow for a much more limited class of prover strategies, though they are able to prove much stronger zero-knowledge results.

<sup>9</sup>Another model, proposed by Babai[Bab], makes all of the verifier's coin tosses public.

the fact that the statement in question is true. The difficulty lies in precisely defining what it means for someone to have learned nothing. This is done by using the notion of simulation. First, we precisely define what we are trying to simulate. If some party,  $A$ , takes part in a protocol, we can define its *view* to be equal to the sequence of messages sent to it, along with its sequence of random coin tosses. For the case of interactive proof systems, we denote by  $\text{VIEW}_V(x)$  the distribution of views verifier  $V$  obtains during the proof that  $x \in L$ .

Now, suppose that some entity,  $S$ , called a simulator, can output a distribution that is “indistinguishable” from party  $A$ ’s view of the protocol. Then, it seems intuitively plausible that  $A$  obtains no more information by running the protocol than he could obtain with the information and resources available to  $S$ . We say that an interactive proof system is zero-knowledge if for any “reasonably powerful” verifier  $\hat{V}$  (possibly malicious), there exists a “weak” simulator,  $M_{\hat{V}}$ , such that  $M_{\hat{V}}(x)$  computes a distribution that is “indistinguishable” from  $\text{VIEW}_{\hat{V}}(x)$ .

### Notions of indistinguishability.

While this is the correct intuition, we still have a lot of blanks to fill in before we can start making well-defined definitions. In the GMR model, the simulators and verifiers are probabilistic polynomial-time Turing machines, and the distributions of  $M_{\hat{V}}(x)$  and  $\text{VIEW}_{\hat{V}}(x)$  are *computationally indistinguishable* by any probabilistic polynomial-time judge. That is, any probabilistic polynomial-time judge cannot distinguish the two distributions with probability greater than

$$\frac{1}{2} + \frac{1}{|x|^c},$$

for any constant  $c$ . This scenario is technically very difficult to work in, but it does allow for some nontrivial theorems to be proven in the standard domain. this notion of zero-knowledge is known as *computational zero-knowledge*.

In our thesis, we actually work with a much stronger notion of indistinguishability, called *statistical indistinguishability*. We consider two families of distributions,  $D_1(k)$  and  $D_2(k)$ . Here,  $k$  is some parameter known as the security parameter. We consider judges that are allowed to observe only  $k^c$  samples from the two distributions, for some constant  $c$ , but are otherwise computationally unbounded. We say that  $D_1(k)$  and  $D_2(k)$  are statistically

indistinguishable if any such judge cannot distinguish  $D_1(k)$  and  $D_2(k)$  with probability greater than

$$\frac{1}{2} + \frac{1}{k^{c'}},$$

for any constant  $c'$ , as  $k$  grows sufficiently large. We can think of  $k$  as a measure of how much security we want from our protocols. Virtually all of our protocols will have a security parameter. For our interactive proof systems, we will often implicitly use a security parameter  $k$  that is equal to  $|x|$ .

The notion of zero-knowledge we will use is known as *statistical zero-knowledge*. In statistical zero-knowledge, we require that the distributions of  $M_{\hat{V}}(x)$  and  $\text{VIEW}_{\hat{V}}(x)$  be statistically indistinguishable. That is, no judge that is allowed to take  $|x|^c$  samples can distinguish  $M_{\hat{V}}(x)$  from  $\text{VIEW}_{\hat{V}}(x)$  with probability greater than

$$\frac{1}{2} + \frac{1}{|x|^{c'}},$$

for any fixed  $c$  and  $c'$ , as  $|x|$  grows sufficiently large.

### 1.3.3 Secure Protocols.

Another explosive area in modern cryptography has been that of secure protocols. In the most general scenario, we consider  $n$  parties,  $P_1, \dots, P_n$ , where  $P_i$  has some secret, which we denote by  $s_i$ . We also have functions  $F_1, \dots, F_n$ , each of which take  $s_1, \dots, s_n$  as arguments. The goal of secure multi-party computation is to allow Player  $P_i$  to learn  $F_i(s_1, \dots, s_n)$ , and nothing else.

A much simpler scenario, which is all we consider in this thesis, is secure two-party computation, first introduced by Yao[Y]. In this scenario, we have two players,  $P_1$  and  $P_2$ , with secrets  $s_1$  and  $s_2$ , and a single function  $F$ . Our goal is for  $P_2$  to learn the value of  $F(s_1, s_2)$ , and nothing else, and for  $P_1$  to learn nothing. Conceptually, we can consider the following physical solution: We have a trusted servant, who is known for discretion and computational ability. The two players give their secrets to the servant, who then computes  $F(s_1, s_2)$ . The servant whispers this value to  $P_2$ , who then shoots the servant dead. Our goal is to simulate this physical scenario without bloodshed.

Surprisingly, more progress has been made for the general problem than for the specific one. Goldreich–Micali–Wigderson [GMW] developed a cryp-

tographic solution for the multi-party problem, under the assumption that the majority of the participants obey the protocol and don't collude afterwards. A series of papers by Ben-Or-Goldwasser-Wigderson[BGW], Chaum-Crépeau-Damgård[CCD], and Ben-Or-Rabin[BR] culminated in a noncryptographic protocol that worked if a majority of the players were honest. Unfortunately, these results say little about the two-party case. They require that less than  $\lfloor n/2 \rfloor$  of the participants are trying to get illicit information, which is 0 for the case of  $n = 2$ . As we will see, there are fundamental reasons for why these solutions break down in this case.

### Bounding knowledge with simulations.

Before we discuss solutions to the two-party problem, we should say precisely what a “solution” is. We wish to satisfy a set of constraints as to what information each party should or should not receive. It is reasonably clear what it means to learn a value of a function, but less clear is what it means to learn “nothing else.” To state this precisely, we fall back on the notion of simulation we used in our discussion of zero-knowledge proofs.

Our notion of what it means for  $P_1$  to learn nothing is the same as with zero-knowledge proofs. In fact, with our protocols,  $P_1$ 's view of the conversation will be easily simulated in probabilistic polynomial time. The difficulty in all our protocols will be in showing that  $P_2$  learns nothing.

Suppose that player  $P_1$  obeys the protocol, using secret  $s_1$ , and a possibly malicious second player,  $\hat{P}_2$ , disobeys the protocol. We can then speak of the malicious player's view of the conversation, which we denote by

$$\text{VIEW}_{\hat{P}_2}(s_1).$$

Now, we can consider a simulator,  $S$ , who runs in probabilistic polynomial time. We would like for  $S$  to simulate an indistinguishable approximation of  $\text{VIEW}_{\hat{P}_2}(s_1)$ .

In the zero-knowledge scenario, we only allowed  $S$  to know the value of  $x$ , the string that was in the language. In this scenario, we want to ensure that  $S$  has access to only the information that  $P_2$  could have obtained by running the ideal protocol. To make this formal, we describe our simulator as talking to  $\hat{P}_2$ , and running the ideal protocol with  $P_1$ . At some point in its conversation with  $\hat{P}_2$ ,  $S$  is allowed to input some value,  $s_2$ , to the ideal protocol, and receive the value of  $F(s_1, s_2)$ . Based on this value,  $S$  must

then finish the conversation with the verifier. We ask that  $\hat{P}_2$ 's view of his conversation with  $S$  be indistinguishable from his view of his conversation with  $P_1$ . In the main body of this thesis, I give players  $P_1$  and  $P_2$  the names Alice and Bob, respectively. As well as instilling a modicum of humanity to a rather dry field, it also allows us to use gender based pronouns without any ambiguity.

### Solutions to the two-party problem.

The first solution to the two-party problem is due to Yao [Y]. This solution was based on the assumption that factoring is hard. It worked for all functions  $F$  which are computable by polynomial-sized circuits. This is essentially as general as one can get, since otherwise it would be hard to compute  $F(s_1, s_2)$  regardless of the privacy constraints. For reasons which will be perhaps clearer after reading this thesis, I prefer the term oblivious circuit evaluation. This term better calls to mind the fact that a circuit is being evaluated. I use the word oblivious to emphasize an important link this problem has to a protocol known as oblivious transfer, as will be made clear later in the introduction.

Yao's notion of indistinguishability was computational. Metaphorically, Alice and Bob could get information they weren't supposed to get, but it would be so scrambled, that any probabilistic, polynomially bounded Turing machine would be unable to distinguish this encrypted information from encrypted noise.

The assumption that factoring is hard was subsequently weakened by Goldreich-Micali-Wigderson [GMW] (based on ideas from [EGL]), to the existence of trapdoor permutations. One might naively hope that even this weakened intractability assumption might be eliminated entirely, as in the multi-party case. However, it is a fairly easy exercise to show that no mental, purely information theoretic solution exists. If we allow our parties to have unbounded computational resources, and only allow them to talk over the telephone, then it is impossible to insure that the two players learn precisely what they are supposed to learn, and nothing else.

## 1.4 Reducing two-party protocols to oblivious transfer.

The second part of my thesis describes a round efficient reduction from two-party protocols, as defined above, to a simple two-party protocol known as oblivious transfer. Oblivious transfer, which we abbreviate as OT, can be described as follows. Alice inputs some bit  $b$  to the oblivious transfer protocol. With probability  $\frac{1}{2}$ , bit  $b$  is sent to Bob. Otherwise, Bob receives some null value, which we denote by  $\#$ .

### 1.4.1 A brief history of oblivious transfer.

Some history behind oblivious transfer is in order. Oblivious transfer was first developed by Rabin [R2]. Rabin gave an implementation for oblivious transfer, based on the assumption that factoring was hard, and that neither party would violate the protocol. In this protocol, Bob would always get information about  $b$ , but half the time this information would be scrambled in such a way that the only way Bob could recover  $b$  would be to factor a large product of two primes.

Fischer–Micali–Rackoff–Tompa–Wittenberg [FMRTW] were able to eliminate Rabin’s assumption that both parties obey the protocol. They use the same complexity assumption as does Rabin, namely that factoring is hard.

Even, Goldreich, and Lempel [EGL] finally reduced the complexity assumption from the intractability of factoring to the existence of trap-door permutations. Their protocol actually allowed them to implement a somewhat stronger protocol, known as “1 out of 2” oblivious transfer (we abbreviate this as  $\frac{1}{2}$ -OT). In this protocol, Alice has two bits,  $b_0$  and  $b_1$ . Bob is allowed to select one of these bits,  $b_x$ , and learn its value. However, Bob is not allowed to learn  $b_{1-x}$ , and Alice is not allowed to learn  $x$ . It is a simple exercise to implement ordinary OT with  $\frac{1}{2}$ -OT.

This thesis continues a long line of research into application of oblivious transfer. Blum [Bl] applied oblivious transfer to coin flipping over the phone, secret-key exchange, and certified electronic mail. Even, Goldreich, and Lempel applied  $\frac{1}{2}$ -OT to contract signing. Brassard, Crépeau, and Robert [BCR] showed that a multi-bit version of  $\frac{1}{2}$ -OT, where Alice has strings  $S_0$  and  $S_1$  instead of bits  $b_0$  and  $b_1$ , could be reduced to  $\frac{1}{2}$ -OT oblivious transfer. Crépeau

[C] later reduced  $\frac{1}{2}$ -OT to ordinary OT.

Oblivious transfer also appears prominently in the implementation of secure two-party protocols. Both of the above protocols for oblivious circuit evaluation use the multi-bit version of  $\frac{1}{2}$ -OT, which can be reduced to simple OT. Unfortunately, these protocols did not constitute a reduction to OT; cryptographic protocols are used to augment the power of OT.

In some very interesting independent work, Goldreich and Vainish [GV] reduced oblivious circuit evaluation to OT, for the case where both parties are honest, i.e., where both players agree to abide by the protocol. Their reduction is extremely simple, and very efficient in the total amount of communication used. There is a catch here, namely that a malicious player can easily steal extra information by violating the protocol. Whereas I go through some difficulties in order to make my protocols robust against active attacks, Goldreich and Vainish use cryptographic mechanisms to strengthen their protocol's security. By using this combination of cryptographic tricks with a weak reduction to OT, they were able to improve on the efficiency of the cryptography based solutions.

### 1.4.2 The results of Chapters 3-5.

The work on oblivious transfer that has been presented in this thesis comprises two main areas. In Chapter 3, we show how to commit and decommit bits, using oblivious transfer, and how to prove zero-knowledge proofs about these committed bits. The novelty of these schemes is that they are extremely round efficient. In order for Alice to commit or decommit a set of bits to Bob, along with a zero-knowledge proof of some assertion about these bits, it is only necessary for her to send a single message to Bob. Furthermore, Bob doesn't have to send any messages to Alice.

This result is of technical interest, since it is not immediately obvious how to do this. However, the reason I like this result is philosophical and literary. The process of proving a statement is not inherently interactive, but some interaction (albeit very bounded) is needed to give a zero-knowledge proof of virtually an nontrivial statement. The introduction of some interaction seems inherently necessary in the usual model for zero-knowledge. I find it interesting that a simple source of entropy can replace this need for interaction. A process that randomly destroys bits of a message can obliterate all of the information for why a theorem is true, yet still leave a cheshire-cat grin

that is a zero-knowledge proof.

From the literary angle, there has got to be a science fiction story here. A long dead race of beings can leave in their rubble clear proof that they were great mathematicians, possessing proofs of Fermat's last theorem, ERH, and the like. Yet because of all the decay, all their delicate arguments have been obliterated. Our credit-greedy hero marches off into the double sunset, in a futile quest for understandable, plagiarizable fragments of these awesome proofs. Maybe not an *Omni* story line, but Isaac Asimov's magazine might be sold on it.

In Chapter 4, I reduce oblivious circuit evaluation to OT, for the special case where  $F$  is computable by an  $NC^1$  circuit. This solution only uses a constant number of rounds of communication. I find this special case very interesting because I can't do it for the general case. For arbitrary circuits, we can still use only a constant number of rounds, but the size of our messages will then be exponential. It is still open whether the general problem can be efficiently solved.

In Chapter 5, I use the results of Chapter 4 to give a reasonably round efficient solution to the general problem of oblivious circuit evaluation. To do this, I develop techniques for chaining together a series of computations in a secure manner. These techniques have proven of independent value in the study of multi-prover interactive proof systems (Chapter 6), and in some more recent work of myself and others on secure protocols.

### 1.4.3 Why all the pain?

Chapter 3 is undoubtedly the most tortuous chapter in this thesis. It comprises about 60 pages, much of it very technical. Chapters 4 and 5 are much more palatable, yet still are rather dense. It is, in retrospect, rather amazing how long it took to write out the details of a few simple ideas. I attribute this to two main causes: the desire for rigor, and the lust for power.

I have attempted to write rigorous proofs for these results. This has been hampered by a blind spot in the cryptography community. Too little effort has gone into how to rigorously prove zero-knowledge results. The most basic protocols, describable in a few minutes, require fairly long and messy proofs. The more intricate protocols have thus far been based on appeals to intuition rather than any rigorous foundation. This thesis is, I feel only typical of the messiness that our current state of the art in proof technology requires. This

messiness will probably be seen again and again, as results make their way from the conference room to the journal paper. Those that don't die along the way, that is.

I have also been excessively neurotic in proving as much as I can prove, regardless of the cost in proof complexity. It is in fact possible to prove somewhat weaker results in a less complicated fashion. Nevertheless, I have for the most part tried to prove the strongest results I know, regardless of the complexity of the machinery involved.<sup>10</sup> I feel that techniques are often more interesting than theorems, so I am showcasing my strongest techniques.

So, to the reader, I give my most sincere sympathy. It is hard to read; it was damn hard to write.

## 1.5 Multi-party interactive proof systems.

The final topic I consider is the study of a variant on interactive proof systems. This work was done by Ben-Or, Goldwasser, Wigderson, and myself [BGKW]. As with the chapter on primality testing, I have been able to include most of the introductory material in the chapter itself. Hence, this introduction will also be brief, dwelling on some of the philosophical oddities and amusements I find in this area of study.

The notion of interactive proofs addresses the very notion of what it means to be convinced that an assertion is true. In real life, we believe in something either because we observe it empirically, or because we have "reasoned out" that such and such must be true. Mathematicians have long eschewed the former mechanism, and emphasized purely mental inference.<sup>11</sup> This is not due to intellectual elitism so much as a genuine desire not to be hoodwinked. For instance, if one tests some number-theoretic identity on random examples, and determines that it indeed holds on these examples, one really can't say that it holds for all integers. It might well fail on a very sparse set of numbers, a set that is nearly impossible to hit at random. Thus,

---

<sup>10</sup>A notable exception is in the zero-knowledge results for Chapter 6, where I use a simple argument that proves about 90% of what one can prove using unbounded pages.

<sup>11</sup>This is not entirely accurate, since mathematicians will frequently gather empirical evidence, such as working out small examples or looking at special cases, to formulate their conjectures. However, such vulgarities can be easily excised from the final papers, and usually are.

a mathematician would wisely be unimpressed by such empirical evidence.

The development of randomized decision procedures brought empirical observations back into the picture. With the standard probabilistic tests for primality, one essentially reasons, "If the number I'm testing was not prime, then I would have been extremely likely to have observed a certain phenomenon. However, I didn't observe this phenomenon, so I'm willing to bet that the number I'm testing is prime." This is indeed a case of reasoning based on empirical observations, despite the fact that the observed phenomenon isn't physical.

To drive home the difference between this type of reasoning and normal mathematical discourse, consider the following point. In order to be rigorous about such reasoning, one must make sure that one's random number generator is functioning properly, just as any other scientist must check out his set of laboratory instruments. When one reads a standard mathematical argument, one does not worry that a fly may have flown into the mathematician's computer when the paper was being written. However, in the type of argument discussed above, flies in the equipment are a perfectly legitimate concern.

Interactive proof systems also bring empiricism back into the picture. The verifier is essentially reasoning that, "If the statement being proven was not true, then in my conversation with the prover, I would have been extremely likely to have observed a certain phenomenon. However, I didn't observe this phenomenon, so I'm willing to bet that the statement is true." As with randomized algorithms, one is placing trust in one's random number generator.

One might argue that we are just saying something familiar in a new way. We can certainly cast a standard proof in this framework. The phenomenon that is observed in the above scenario would simply be the prover sending anything but a correct proof of the statement in question. Still, when we consider the standard interactive proof for graph nonisomorphism, for instance, it is clear that something fundamentally different is going on. There is often no apparent mapping from the reasoning the verifier goes through in the interactive proof to any normal mechanism for proof verification.

I think this point is brought home most clearly by the existence of statistically zero-knowledge proofs. At the end of the protocol, the verifier has no information that he didn't have before. Thus, the proof could not have helped the verifier make any verification based on purely mental inference.

The entire reason the verifier has any confidence that the theorem is being proved is due to reasoning about empirical observations.

I don't think we can honestly say the same about computational zero-knowledge proofs. Typically, these proofs do give a great deal of information about why the statement in question is true, albeit in encrypted form. Given that the set of statements that can be proven in statistical zero-knowledge seems to be a very small subset of what can be proven interactively, it seems that giving actual information is necessary for many interactive proofs. One might conjecture that in any reasonable type of probabilistic proof system, there are assertions which can be "proven" only by giving a lot of information to the verifier.

What we were able to show in our multi-prover model was a scenario in which anything provable could be provable without transferring any factual information whatsoever. The verifier simply asks questions of the provers, observes their reactions, and check to see if certain phenomena occurs. The provers' answers have nothing to do with why the statement in question is true, but somehow convince the verifier to believe the statement just the same. Whenever the provers seem to need to the verifier some solid information that it could not compute by itself, we can mechanically transform the protocol into one in which this information is no longer given.

On a higher level, the reason we can argue that the transformed protocol makes any sense has something to do with the fact that the original protocol transferred a lot of information. Yet somehow, we are able to argue that information laden behavior on the part of the provers can be replaced by information free behavior. But what has all the information been transformed into? I can push around lemmas and theorems all day, but on some fundamental level, I don't really know.

## Chapter 2

# New Techniques in Primality Testing.

### 2.1 Introduction.

#### 2.1.1 Historical overview, and a brief review of previous work.

The study of prime numbers is one of the oldest fields in mathematics. To put our work in context, we give an admittedly incomplete history of the field.

#### The early days of primality testing.

The problem of distinguishing prime numbers from composites has been with us for well over two thousand years. Eratosthenes[E] came up with the first recorded algorithm for primality testing, in the 3rd century BC. He showing how to efficiently generate the set of primes from 1 to  $N$  in  $O(N \ln \ln N)$  arithmetic steps. This is exceedingly fast, since an average of only  $O(\ln \ln N)$  arithmetic operations are used to test each number for primality. Indeed, this bound is exponentially better than all of the algorithms I will discuss in this chapter, and I do not know of any algorithm that is substantially faster than his. Eratosthenes' paper represents the golden age of primality testing.

Of course, Eratosthenes was cheating a little bit. Given the first  $N$  numbers, it is easy to eliminate approximately  $N/2$  of them as candidates for

primality, namely the even numbers greater than 2. These numbers may be eliminated in a total of  $O(n)$  steps, for an average of  $O(1)$  computations per number. Likewise, one can also cross off multiples of 3, 5, 7, 11, ... in a relatively efficient manner. Thus it should really be no surprise that the average amount of computations can be so low; only a very sparse set of numbers require many calculations to be revealed as composite.

There are two ways natural problems related to Eratosthenes' problem of generating the primes from 1 to  $N$ . First, how can one efficiently determine if a single  $N$  is prime? Second, how quickly can one generate a large prime number? The subtlety of these problems becomes clear when one considers values of  $N$  which are 100 digits long. If you were to try the elementary school algorithm, namely dividing  $N$  by all numbers in the range  $[2, \sqrt{N}]$ , you would die.

Eratosthenes no doubt proposed these problems to his graduate students, and would have cheerfully signed their dissertations had any of them met with any success. Unfortunately, none did, and the fragile chain of scholarship was broken. The Roman Empire crumbled, and the dark ages set in.

### **The modern age of primality testing.**

Finally, starting in the 17<sup>th</sup> century, mathematicians (Fermat, Euler, Legendre, and Gauss, to name a few) began to study primality once more. Their work laid the foundation for a new age in primality testing, which began in the 1970's. Using elementary results from number theory, specifically results on quadratic residuosity and the structure of  $Z_n^*$ , Miller, Solovay-Strassen, and Rabin developed the first efficient algorithms for these problems.

Solovay-Strassen [SS] came up with the first practical primality test. Their algorithm is randomized, i.e. on input  $N$  it will flip a series of coins, and compute its answer based on  $N$  and this sequence of coin flips. If it is given an input  $N$ , which is composite, it will with high probability output a proof that  $N$  is composite. If it is given a value of  $N$  which is prime, it will fail to come up with anything interesting at all, which is very interesting indeed. The fact that it did not come up with a proof of compositeness gives probabilistic support to the assertion that it is prime. Thus, on input  $N$ , the Solovay-Strassen algorithm outputs either

- “ $N$  is composite, and here is the proof ...,” or,

- “ $N$  is prime, I think. If it’s composite, then I’ve sure been unlucky.”

Note that when the algorithm outputs declares a number composite, it is definitely composite. It is only in declarations of primality that the algorithm hems and haws.

Miller [M] developed a deterministic polynomial-time algorithm for primality testing based on the Extended Riemann Hypothesis (ERH). On input  $N$ , the algorithm searches for a proof that  $N$  is composite. If it finds one, it stops and reports that  $N$  is composite, along with its proof of compositeness. If it doesn’t find a proof of compositeness, the algorithm reports that  $N$  is prime. Miller showed that if ERH holds, then his algorithm would always find a proof of compositeness for any composite number. Thus, the output of this algorithm is of the general form,

- “ $N$  is composite, and here is the proof . . .,” or,
- “If ERH is true, then  $N$  is prime.”

As with the Solovay–Strassen tests, whenever Miller’s test declares compositeness, the statement is unconditional. When it states that a number is prime, it still allows for the possibility of error. It does not blame a mistake on bad luck, but to a violation of ERH. Unfortunately, while most mathematicians believe that ERH is true, the proof of this fact may quite possibly require 21<sup>st</sup> century mathematics.

Rabin[R] analyzed a very natural probabilistic variant of Miller’s algorithm, and showed that this procedure had the same properties as described for the Solovay–Strassen algorithm.

In this same mathematical vein, much more complicated deterministic algorithms have been developed by Adleman–Pomerance–Rumely[APR], and Cohen–Lenstra[CL]. These algorithms are deterministic, and do not rely on any unproven assumptions. However, they are relatively slow, requiring  $n^{\theta(\ln \ln n)}$  steps on an input  $N$  of length  $n$ . Furthermore, they do not provide any succinct proof of the primality number of a number it declares prime. Thus, one could not, after running their algorithm, quickly convince someone else that the number was prime.

### 2.1.2 The problem of prime certification.

The probabilistic algorithms of Solovay–Strassen and Miller–Rabin share a common deficiency. They are very good at proving that a number is composite, but they have no capability for proving a number prime. This would not be a handicap if they could always be guaranteed to prove compositeness on composite inputs. However, such a guarantee cannot be made for any probabilistic algorithm. There is always a finite probability that, whenever the algorithm chooses a random bit, it will always get the value 0. If the algorithm found proofs of compositeness all the time, it would do so in this case. But then one could trivially make the algorithm deterministic by replacing its random tape with all zeros.

Thus, the probabilistic algorithms which represent the previous state of the art are inherently asymmetric. They are not so much primality testers as compositeness provers. The natural question, then, is whether one can do for prime numbers what one can do for composite numbers: Can one produce short proofs of a number's primality that can be verified by a deterministic, polynomial-time bounded Turing machine. We call such a short proof a *certificate of primality*. We devote most of this chapter to this problem. A very interesting subproblem is how to efficiently generate large primes which have short certificates of primality. This we handle as a simple corollary of our techniques.

#### Pratt's contribution: Short certificates exist.

Unlike the case with composite numbers, it is not at all obvious that short certificates of primality even exist. Given a composite number, one can prove that it is composite by exhibiting one of its nontrivial divisors. Such a proof may be very hard to come by, but it is short and easily verified as correct. Thus, composites are in  $NP$ . However, this seems to give little insight in how one can, even with infinite power at one's disposal, come up with a short proof that no such divisor exists.

Somewhat surprisingly, Pratt [Pr] has shown that such a short certificate of primality always exists (i.e. primes are in  $NP$ ). His proof relies on the fact that a number  $n$  is prime iff there exists an element  $a \in Z_n^*$  of order  $n - 1$ . This is a simple fact by the standards of modern number theory, but is vastly more subtle than the corresponding fact needed to show that composites are

in  $NP$ . While the simple proof that a number is composite gives no insight into how to efficiently prove a number composite, it turns out that Pratt's ideas are of use in efficient prime certification, albeit in substantially more general form.

### 2.1.3 The results of this chapter.

In this chapter, we present a new, simple methodology for applying group theory to the problem of prime certification. We use this methodology, in conjunction with the theory of elliptic curves, to develop an algorithm for prime certification. This algorithm has the following three properties.

1. Given inputs of length  $k$ , our algorithm produces certificates of primality that are of length  $O(k^2)$ , and which require  $O(k^4)$  steps to verify.
2. Our algorithm terminates in expected polynomial time on every prime number if the following conjecture is true:

$$(\exists c_1, c_2 > 0) \pi(x + \sqrt{x}) - \pi(x) \geq \frac{c_2 \sqrt{x}}{\log^{c_1} x},$$

for  $x$  sufficiently large. Here,  $\pi(n)$  denotes the number of prime numbers which are less than  $n$ . This conjecture is very believable, for reasons which will be discussed later.

3. Suppose that the algorithm is run on inputs of length  $k$ . There exists a constants  $c_1, c_2$  such that for all  $k$  sufficiently large, the algorithm will terminate in expected  $c_1 k^{11}$  time for all but at most,

$$2^k / 2^{k^{\frac{c_2}{11 \ln k}}},$$

of the inputs. In other words, the algorithm can be proved to run quickly on all but a vanishingly small fraction of the prime numbers.

This result has two major corollaries. First, a previously long-standing open question in primality testing is whether there exists an infinite set of primes which can be recognized as prime in polynomial time. Our algorithm shows that not only infinitely many, but most primes can be recognized in expected polynomial time.

Second, we solve the problem of efficiently generating large certified primes. Previous to our work, no method was known which provably produced more than a finite number of certified primes. Since we can certify most primes as prime, we can use the following simple algorithm to generate a  $k$ -bit prime.

1. Uniformly generate a random  $k$ -bit integer,  $n$ . Using a standard probabilistic test, attempt to prove it composite. If the attempt succeeds, repeat Step 1.
2. Using our test, attempt to quickly (using only  $k^c$  steps, for some constant  $c$ ) find a certificate of primality. If this succeeds, output  $n$  with its certificate. Otherwise, go to Step 1.

In other words, we can simply keep on trying numbers until we find one we can quickly certify. The fact that we can certify nearly all primes in expected polynomial time, and the fact that a random  $k$ -bit number will be prime with probability  $O(1/k)$  (the prime-number theorem), ensures that the above algorithm will terminate in expected polynomial time. This distribution will be statistically very close to the uniform distribution.

#### 2.1.4 Techniques used.

Our methodology, in its most general form, is very simple. We will in fact give an exposition of our technique which only uses elementary number theory. However, in order to make efficient algorithms that are amenable to rigorous analysis, we rely on a strong foundation of results and techniques from computer science and number theory. We cite below some of the work that made ours possible.

#### Previous primality tests.

We use the previous state of the art in primality testing, both the randomized algorithms, the deterministic algorithms, and Pratt's proof that primes have short certificates. These three results are used in quite different ways.

Pratt gave a technique whereby given an element  $a \in \mathbb{Z}_p^*$ , of order  $p - 1$ , one could reduce the problem of proving  $p$  prime to that of proving  $q$  prime, for all  $q|p - 1$ . We use this same approach in our methodology. We show

how, using a group  $G$  which is somehow associated with  $p$ , and an element  $a \in G$  of order  $q$ , one can reduce the problem of proving  $p$  prime to that of proving  $q$  prime. Our algorithm uses this trick to make a series of reductions, at each step reducing the number to be proven prime by at least a constant.

Our reduction technique will spit out assertions of the form “ $p$  is prime if  $q$  is.” Most of these assertions will be trivially vacuous, since for most of these assertions,  $q$  will not be prime. In order to figure out what assertions are useful, it is necessary to be able to rapidly determine if  $q$  is prime. To do this, we use the probabilistic tests of Solovay-Strassen and Miller-Rabin.

Finally, in order to prove stronger results about our algorithm, we will make use of the deterministic algorithms of Adleman-Pomerance-Rumely and Cohen-Lenstra. Essentially, this allows us to stop when the number we need to prove prime is sufficiently small compared to the original number we wished to prove prime.

### The theory of elliptic curves.

Given a prime  $p \geq 5$  and a pair  $(A, B)$  where  $A, B \in \mathbb{Z}_p$  and  $4A^3 + 27B^2 \not\equiv 0 \pmod{p}$ , we consider solutions  $(x, y)$  to the equation

$$y^2 \equiv x^3 + Ax + B \pmod{p}.$$

These sets of ordered pairs, when augmented by a extra point  $I$ , are the points of an *elliptic curve* over  $GF(p)$ . There is a natural addition operation, under which the points of an elliptic curve form an abelian group. Elliptic curves have been studied extensively from the standpoint of pure mathematics, and have been recently used in the development of algebraic algorithms.

Schoof [Sc] uses elliptic curves in an algorithm for computing square roots of small integers  $x \pmod{p}$ . His algorithm is polynomial in  $x$  and  $|p|$ . As part of this work, he shows how to compute the order of a group generated by an elliptic curve over  $GF(p)$ , in deterministic polynomial time. This algorithm is crucial to our primality test.

Lenstra [L] uses elliptic curves to obtain an integer factorization algorithm which uses nearly constant memory, and whose running time is a function of the size of the smallest prime divisor of the integer to be factored. The running time analysis of Lenstra’s algorithm depends on a very plausible assumption concerning the distribution of smooth numbers in small intervals, and requires no unproven assumptions about elliptic curves. This

independence is due to a useful result he obtains concerning the distribution of the orders of elliptic curves. We use his distribution result to eliminate any unproven assumptions about elliptic curves from the the analysis of our algorithm's running time.

### Results on the density of primes in small intervals.

The running-time analysis of our algorithm depends on the frequency of primes in intervals of the form  $[x, x + \sqrt{x}]$ , i.e., on the value of  $\pi(x + \sqrt{x}) - \pi(x)$ . We use results on the asymptotic distribution of prime numbers to bolster our belief in our conjecture, and to allow us to prove our algorithm fast on most inputs.

We recall the conjecture under which our algorithm will work for all primes:

$$(\exists c_1, c_2 > 0) \pi(x + \sqrt{x}) - \pi(x) \geq \frac{c_2 \sqrt{x}}{\log^{c_1} x},$$

for all  $x$  sufficiently large. The Prime Number Theorem states that for sufficiently large  $x$ ,  $\pi(x)$  will approach  $x / \ln x$ . This suggests that our conjecture might be true, with  $c_1 = 1$ . However, this theorem does not directly imply anything about the behavior of  $\pi(x)$  over small intervals. A famous, widely believed conjecture of Cramer states that for sufficiently large  $x$ ,

$$\pi(x + \ln^2 x) - \pi(x) > 0.$$

This directly implies our conjecture, with  $c_1 = 2$ .

While no one has been able to prove our conjecture for all numbers, Heath-Brown [HB] have shown that our conjecture is true most most intervals. One of their technical lemmas implies the following result (communicated to us by Maier and Pomerance [MP]).

**Theorem 2.1** [Heath-Brown] Call an integer  $y$  sparse if there are less than  $\sqrt{y}/2 \lfloor \ln y \rfloor$  primes in the interval  $[y, y + \lfloor \sqrt{y} \rfloor]$ . Then there exist a constant  $\alpha$  such that for sufficiently large  $x$ ,

$$|\{y : y \in [x, 2x], y \text{ is sparse}\}| < x^{5/6} \ln^\alpha x.$$

Heath-Brown's theorem is crucial to the analysis of our algorithm on uniformly distributed input.

### 2.1.5 Subsequent research using our methodology.

Our methodology has been used in two more recent algorithms. First, and foremost, Adleman and Huang [AH] have developed an algorithm that is guaranteed to find short certificates for all prime numbers. To do this, they first sharpen our work with groups generated by elliptic curves. They bound above the fraction of “bad”  $k$ -bit primes, which the elliptic curve based algorithms could not quickly certify, down to  $2^{-\Omega(k)}$ . This by itself is not of great interest, but turns out to be crucial to their next, much larger step. They then apply our methodology to a different class of groups, those generated by hyperelliptic curves.<sup>1</sup> This yields an algorithm which allows them to reduce the proof of primality for a prime  $p$  to a proof of primality for a sufficiently randomized prime  $q$ . They can then run the sharpened version of our algorithm to show that  $q$  is prime. It can be shown that  $q$  is sufficiently random so that it will be certifiable with high probability.

Unfortunately, both our algorithm and the more sophisticated algorithm of Adleman-Huang are incredibly slow in practice. Our algorithm takes  $O(n^{11})$  expected time on most primes, and their algorithm is even worse. However, Atkin [At] has a variant of our method, also using groups generated by elliptic curves, that runs much quicker in practice. Using this algorithm, primes with hundreds of digits can be routinely certified. Unfortunately, the mathematically sophisticated modification necessary to improve the algorithm’s running time has frustrated attempts at rigorous analysis. An algorithm which is provably fast (in the practical sense of the word) still eludes us.

### Outline of the Chapter.

In Section 2.2, we develop our methodology for primality proving. In Section 2.3 we give a quick introduction to elliptic curves, their algebraic structure over  $Z_p$  and  $Z_n$ , and show how they fit into our methodology. In Section 2.4, we give our primality proving algorithm. In Section 2.5, we show that our algorithm produces certificates for all primes in expected polynomial

---

<sup>1</sup>So easy to say, so hard to do. This application involved some of the most impressive algebraic algorithms work that I have ever seen. Whereas we could make use of a large body of theorems and algorithms, they had to build much of their machinery largely from scratch.

time, modulo a number-theoretic conjecture. We then extend this argument to show that our algorithm produces certificates for almost all primes in expected polynomial time. This last theorem depends on no unproven assumptions.

## 2.2 Our primality proving methodology.

In this section, we present our primality proving methodology. Essentially, it generalizes a very simple, easily proved primality criterion. We first give the easy criterion, and then show how to extend to more abstract structures.

### 2.2.1 A primality criterion from elementary number theory.

Our methodology starts with the well-known fact that if  $p|n$ , then  $Z_n^*$  has a natural projection to  $Z_p^*$ , namely that obtained by taking the element of  $Z_n^*$  modulo  $p$  in the obvious way. This projection, which we denote by  $\rho_p^n : Z_n^* \rightarrow Z_p^*$ , preserves multiplication. That is, if  $a, b, c \in Z_n^*$ , and  $c = ab$ , then  $\rho_p^n(c) = \rho_p^n(a)\rho_p^n(b)$ . For example, if  $n = 15$  and  $p = 5$ , then  $11 = 8 \cdot 7 \bmod 15$  implies that  $1 = 3 \cdot 2 \bmod 5$ . As a corollary, the projection also preserves the identity element, i.e.  $\rho_p^n(1) = 1$ . For notational convenience, we adopt the convention of denoting  $\rho_p^n(x)$  by  $x_p$ , and  $\rho_p^n(x_i)$  by  $x_{ip}$ , whenever  $n$  is understood.

We can use the above fact, along with the fact that  $|Z_p^*| < p$ , to derive a simple primality criterion.

**Theorem 2.2** Suppose that for some  $a \in Z_n^*$  and integer  $q > \sqrt{n}$ , we have  $(a - 1, n) = 1$ , and  $a^q \equiv 1 \bmod n$ . Then if  $q$  is prime,  $n$  is also prime.

**Proof:** Our proof is by contradiction. Suppose that some prime  $p \leq \sqrt{n}$  divides  $n$ . Then, since our projection preserves multiplication, we have

$$a_p^q \equiv 1 \bmod p.$$

Clearly, the order of  $a_p$  divides  $q$ . If  $q$  is prime, then the order of  $a_p$  is either 1 or  $q$ . However, the order of  $a_p$  is clearly at most  $p - 1 < q$ , so the order of  $a_p$  must be 1. Thus,  $a_p \equiv 1 \bmod p$ . Since  $a \equiv a_p \bmod p$ , we have  $a \equiv 1 \bmod p$ .

We now show that  $a \not\equiv 1 \pmod p$ . This follows from the fact that  $a - 1$  and  $n$  are relatively prime. If  $a \equiv 1 \pmod p$ , then  $p|a - 1$ , so  $p|(a - 1, n)$ , which is a contradiction. ■

### An example.

Here is an example of how we might use our primality criterion to show that a large number was prime.

Let us show that 179 is prime. We first observe that

$$9^{89} \equiv 1 \pmod{179}$$

, and  $(8, 159) = 1$ . The first fact can be confirmed by the standard “repeated squaring” algorithm for modular exponentiation. The second fact can be checked by Euclid’s algorithm. Finally,  $89 > \sqrt{179} = 13.37 \dots$ . Thus, by our primality criterion, 179 is prime if 89 is prime. We have reduced our problem to showing that 89 is prime.

Now we wish to show that 89 is prime. We observe that

$$45^{11} \equiv 1 \pmod{89},$$

$(44, 89) = 1$ , and  $11 > \sqrt{89} = 9.43 \dots$ . Thus, if 11 is prime, then 89 must be prime. But everyone knows that 11 is prime, so we have shown that 179 is prime.

Of course, the above toy theorem is of little practical use in prime certification. The example has been cooked up to work, and in fact would have failed miserably if we had tried to reduce 13 to a smaller prime.

## 2.2.2 The general scenario.

We now show how to generalize this approach to more abstract structures. Let us step back a bit, and see what we really needed to prove the above primality criterion. Essentially, we needed the following three facts.

- The structure of  $Z_n^*$  is such that we can guarantee that if  $p|n$ , there is a multiplication preserving projection from  $Z_n^*$  to  $Z_p^*$ . We can make this guarantee without having any information about the factors of  $n$ , or even whether  $n$  is prime.

- In our structure associated with  $n$ , we can guarantee that our element  $a$  does not project to the identity element of the group associated with  $p$ . In other words, we can guarantee that

$$a_p \not\equiv 1 \pmod{p}.$$

- Our value of  $q$  is guaranteed to be greater than the size of  $Z_p^*$ , for  $p \leq \sqrt{n}$ .

Given these facts, the theorem goes through in a straightforward manner.

Now, let us see where we can start generalizing our assumptions. First, we never used the fact that  $Z_n^*$  is always guaranteed to be a group. All we used was the fact that there is always a multiplication preserving projection from  $Z_n^*$  to  $Z_p^*$ . We did not even use the fact that the multiplication operation was always defined. Second, we did not use any special facts about  $Z_p^*$ . We only used the fact that  $Z_p^*$  was a group whose size had a known upper bound as a function of  $p$ . Thus, we can work with much more general algebraic structures.

The other main generalization we will use is that it is not necessary to associate a single structure,  $Z_n^*$  with  $n$ , and a single group,  $Z_p^*$ , with each prime  $p$ . Instead, we can just as well associate an arbitrary set of structures with  $n$  and an arbitrary set of groups with  $p$ . Before, we guaranteed that our structure associated with  $n$  had a multiplication preserving projection to the group associated with any prime divisor of  $n$ . Now, we require that a structure associated with  $n$  must project to at least one of the groups associated with each prime divisor  $p$  of  $n$ .

With these thoughts in mind, we now give the definitions and primality criterion that underlie our methodology.

### Simple structures with multiplication.

We wish to formalize what it means for one of our simple algebraic structures, which will serve an analogous role to  $Z_n^*$ , to project to a group. First, we must precisely define our simple structures.

**Definition 2.1** We define a *weak structure* to be a triple  $W = (S, I_S, \otimes)$ , where  $S$  is a set,  $I_S$  is some distinguished element of  $S$ , and  $\otimes$  is a partial function from  $S \times S$  to  $S$ .

In other words, we have a set of elements, with a multiplication operation and a distinguished “identity” element. The multiplication operation need not have any special properties, or even be defined for all pairs of elements in  $S$ . As with groups, we will use the notation  $a \in W$  to mean  $a \in S$ .

We wish to speak about exponentiation in this scenario. The meaning of exponentiation is not obvious, since weak structures do not even guarantee that multiplication is well-defined, let alone associative. Hence, it may be, for example, that computing  $a^6 = a^3 \cdot a^3$  will give a different answer than computing  $a^6 = a \cdot a^5$ . Hence, we must define formally what we mean when we say  $a^q$ . For integers  $q > 0$ , we define  $a^q$  by

$$a^q = \begin{cases} a & \text{for } q = 1, \\ (a \otimes a)^{q/2} & \text{for } q \text{ even}, \\ a \otimes a^{q-1} & \text{for } q \text{ odd}. \end{cases}$$

If any of the multiplications or exponentiations given by the definition of  $a^q$  are undefined, then we say that  $a^q$  is undefined. This is just the “repeated squaring” algorithm for exponentiation. By using this algorithmic definition, we are mathematically precise, and ensure that computing  $a^q$  will take only  $O(\log q)$  multiplications.

### Projections from weak structures to groups.

We now define what it means for a weak structure to project to a group.

**Definition 2.2** Let  $W = (S, I_S, \oplus)$  be a weak structure, and let  $G$  be a group with identity element  $I_G$ . We say that a function  $\tau : S \rightarrow G$  *projects*  $W$  to  $G$  if

$$(\forall x, y, z \in W) z = x \otimes y \implies \tau(z) = \tau(x)\tau(y) \text{ and,}$$

$$\tau(I_S) = I_G.$$

Note that a function which maps all the elements of  $W$  to the identity element  $I_G$  is a legitimate projection. It will be useful to be able to say that a projection does not map an element to the identity.

**Definition 2.3** Let  $W$  be a weak structure,  $G$  be a group, and let  $\tau$  be a projection from  $W$  to  $G$ . For  $a \in W$ , we say that  $\tau$  *preserves*  $a$  if  $\tau(a) \neq I_G$ .

The following lemma illustrates the usefulness of projections. It will prove very useful in the proof of our main theorem.

**Lemma 2.1** Let  $W = (S, I_S, \oplus)$  be a weak structure,  $G$  be a group with identity element  $I_G$ , and  $\tau$  be a projection from  $W$  to  $G$ . Let  $a \in W$  and  $q$  be a positive integer such that  $a^q$  is defined. Then  $\tau(a)^q = \tau(a^q)$ .

**Proof:** Our proof is by total induction on  $q$ . For  $q = 1$  the statement is clearly true. For  $q$  even, we have  $a^q = (a \otimes a)^{q/2}$ , and hence

$$\begin{aligned}\tau(a^q) &= \tau((a \otimes a)^{q/2}) \\ &= \tau((a \otimes a))^{q/2},\end{aligned}$$

by our inductive hypothesis (note that all these quantities must be defined, since otherwise  $a^q$  would not be defined). Since  $\tau$  preserves multiplication, we have  $\tau(a \otimes a) = \tau(a)\tau(a) = \tau(a)^2$ . Combining this with the equation above yields

$$\tau((a \otimes a))^{q/2} = (\tau(a)^2)^{q/2} = \tau(a)^q.$$

Similarly, if  $q$  is odd, we have

$$\begin{aligned}\tau(a^q) &= \tau(a \otimes a^{q-1}) \\ &= \tau(a)\tau(a^{q-1}) \text{ (by definition of } \tau) \\ &= \tau(a)\tau(a)^{q-1} \text{ (by inductive hypothesis)} \\ &= \tau(a)^q \quad \blacksquare\end{aligned}$$

We actually use the following corollary of Lemma 2.1, which follows trivially from the lemma, and the fact that  $\tau(I_S) = I_G$ .

**Corollary 2.2** Let  $W, G, \tau, a, q$  be as in Lemma 2.1. If  $a^q = I_S$  then  $\tau(a)^q = I_G$ .  $\blacksquare$

### Generalizing $Z_p^*$ via group assignments.

For our generalization, we wish to be able to associate an arbitrary number of groups to a given prime  $p$ . This motivates the following definition.

**Definition 2.4** we define a *group assignment* to be a function  $\gamma$  that maps prime numbers to sets of finite groups.

For example, one legal group assignment  $\gamma$  is defined by

$$\begin{aligned}\gamma(2) &= \{Z_5^*, Z_7^*\} \\ \gamma(3) &= \{\} \\ \gamma(5) &= \{Z_5^* \times Z_5^*, S_n \text{ (for } n > 1000)\} \\ &\vdots\end{aligned}$$

Note that the sets we define may be infinite or empty, and the groups we use may be nonabelian.

In our methodology, we need to bound the sizes of groups assigned to primes of a given size. Precisely, we will need to bound, for a given  $n$ , the size of the largest group assigned to a prime  $p \leq \sqrt{n}$ . We thus define the function  $\mathcal{S}(n, \gamma)$  by

$$\mathcal{S}(n, \gamma) = \max_{p \leq \sqrt{n}} \max_{G \in \gamma(p)} |G|,$$

where we adopt the convention that  $\max_{G \in \gamma(p)} |G| = 0$  if  $\gamma(p) = \{\}$ .

### Generalizing $Z_n^*, a$ via $(n, a)$ -projecting weak structures.

We now wish to generalize the role of  $Z_n^*$  and element  $a$  in our primality criterion. We do so through the notion of  $(n, a)$  projection.

**Definition 2.5** Let  $\gamma$  be a group assignment, and let  $W$  be a weak structure. Let  $a \in W$ , and let  $n$  be some positive integer. We say that  $W$   $(n, a)$ -projects if, for all prime  $p$  such that  $p|n$ , there exists a group  $G_p \in \gamma(p)$  and a mapping  $\tau_p$  that projects  $W$  to  $G_p$  while preserving  $a$ .

In other words, a weak structure  $W$  has a natural projection to a group associated with every prime divisor of  $n$ . These projections are not degenerate at  $a$ ; they project  $a$  to something other than the identity.

## 2.2.3 The general primality criterion.

Using the above terminology, we can succinctly give our general primality criterion.

**Theorem 2.3** Let  $\gamma$  be a group assignment, and let  $W = (S, I_S, \oplus)$  be a weak structure. Suppose the following two conditions hold.

1. There exists an element  $a \in W$ , and an integer  $q$  such that  $a^q = I_S$ , and  $q > \mathcal{S}(n, \gamma)$ .
2.  $W$   $(n, a)$ -projects.

Then if  $q$  is prime,  $n$  will also be prime.

**Proof:** The proof of Theorem 2.3 follows the proof of Theorem 2.2. We use proof by contradiction. Suppose that  $n$  was composite. Let  $p$  be a prime such that  $p|n$ ,  $p \leq \sqrt{n}$ . By the conditions of the theorem, and the definition of  $(n, a)$  projection, there must be a group  $G \in \gamma(p)$ , and a mapping  $\tau$  that projects  $W$  to  $G$  while preserving  $a$ . Furthermore, by Corollary 2.2, we have  $\tau(a)^q = I_G$ , where  $I_G$  is the identity element for  $G$ .

We now derive a contradiction from  $\tau(a)^q = I_G$ . By elementary group theory, the order of  $\tau(a)$  must divide  $q$ . Since  $q$  is prime, the order of  $\tau(a)$  must therefore be either 1 or  $q$ . If the order of  $\tau(a)$  was 1, then we would have  $\tau(a) = I_G$ , which contradicts the fact that  $\tau$  preserves  $a$ . However, since  $q > \mathcal{S}(n, \gamma)$ , we have, in particular,  $q > |G|$ . Therefore, the order of  $\tau(a)$  cannot be  $q$ , since the order of a group element can't be larger than the size of the group. The theorem follows. ■

It is instructive to see how the elementary primality criterion is a special case of Theorem 2.3. First we let group assignment  $\gamma$  be defined by

$$\gamma(p) = \{Z_p^*\}.$$

Since  $|Z_p^*| = p-1$ , we have  $\mathcal{S}(n, \gamma) < \sqrt{n}$ , so our  $q > \sqrt{n}$  condition translates into  $q > \mathcal{S}(n, \gamma)$ . Next, we let our weak structure  $W$  be  $Z_n^*$ . The condition that  $(a-1, n) = 1$  implies that  $a \not\equiv 1 \pmod{p}$  for  $p|n$ . This implies that  $W$  will  $(n, a)$ -project, using the trivial mapping from  $Z_n^*$  to  $Z_p^*$ . Thus, we can easily fit the old primality criterion into the new framework.

## 2.2.4 Translating notation from multiplicative to additive groups.

The arguments of this section use the multiplicative rather than the additive notation for groups. This was motivated by our example, since  $Z_p^*, Z_n^*$  are

multiplicative groups. However, for other groups, particularly the groups we will focus on, the additive notation is used. This is a matter of convention, and is of no mathematical consequence; whether one calls the group operation multiplication, addition, or funny-op, it is still just a group operation. Thus, our notions of projection, and our various theorems all make sense in the additive notation.

It is fairly easy to make the notational transition between this section and the next. First, wherever the above theorems refer to multiplication, one simply translates them to refer to addition. Thus,  $ab$  would translate to  $a + b$ . Second, wherever exponentiation is used, simply replace by multiplication. Thus,  $a^q$  translates to  $qa$ .

## 2.3 Applying our methodology to elliptic curves.

In this section, we show how to apply our methodology to the groups generated by the points on elliptic curves. This will allow us to, using our general primality criterion, formulate a primality criterion based on elliptic curves.

For those unfamiliar with the basic theory of elliptic curves, we present a brief introduction to this field. To get a more complete background on this subject, we refer the reader to a textbook on the subject by Silverman [Si], and a survey paper by Tate [T]. The “classical” results discussed in this chapter may all be found in these references.<sup>2</sup>

### 2.3.1 What is an elliptic curve?

First, let us define an elliptic curve, represented in Weierstrauss normal form.

**Definition 2.6** Let  $\mathcal{F}$  be a field whose characteristic is not 2 or 3.<sup>3</sup> An elliptic curve is an ordered pair  $(A, B)$ , where  $A, B \in \mathcal{F}$ , and  $4A^3 + 27B^2 \neq 0$ .

---

<sup>2</sup>The more technically sophisticated reader is also encouraged to read A. Lenstra and H. Lenstra’s manuscript on factoring and primality testing [LL]. We are deeply indebted to this exposition for clarify the subject to us, and substantially contributing to the exposition in this chapter.

<sup>3</sup>That is,  $2, 3 \neq 0$  in the field. this will hold for all the fields we will consider in this chapter.

This definition is formally correct, but not very revealing. To give an idea of what we mean by an elliptic curve, we define the points on an elliptic curve.

**Definition 2.7** Let  $\mathcal{F}$  be a field whose characteristic is not 2 or 3, and let  $(A, B)$  be an elliptic curve over  $\mathcal{F}$ . We define the points of  $(A, B)$  to be the set of ordered pairs  $(x, y)$  such that  $y^2 = x^3 + Ax + B$ , and an additional element,  $I$ . We denote these points by  $E_{A,B}(F)$ . If  $F = Z_p$ , we use the abbreviation  $E_{A,B}(p)$  to denote  $E_{A,B}(Z_p)$ .<sup>4</sup>

The additional element  $I$  is often called “the point at infinity.” It will serve as the identity element when we impose a group structure on the points of an elliptic curve. It may seem cumbersome to have a point whose form is completely different from all the other points, and indeed there exist notations for elliptic curves which are more uniform. However, it turns out that  $I$  will play a very important role in our future discussions, and thus the notation we choose is very convenient.

### 2.3.2 The group structure of points on an elliptic curve.

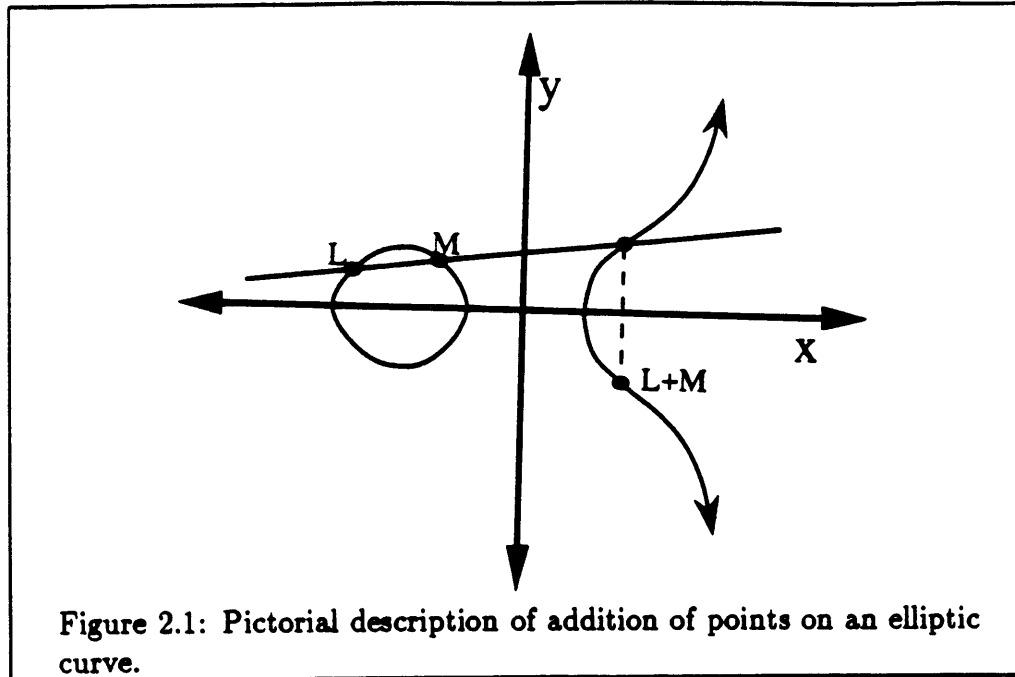
There exists a natural way of defining addition for the points on an elliptic curve. First, we make  $I$  the identity, by defining  $a + I = I + a = a$ . For the rest of this discussion, we assume that the points we wish to add can be written as  $L = (x_1, y_1)$  and  $M = (x_2, y_2)$ .

When our base field consists of the reals, then we can interpret our addition operation as illustrated in Figure 2.1. Given two distinct points,  $L$  and  $M$ , we first consider the line connecting  $L$  and  $M$ , and locate the third intersection point of the line with the points on curve  $(A, B)$ . We then reflect this third point over the  $x$ -axis, and define the resulting point as  $L + M$ .

We must address a few technical points. First, if  $L = M$ , we can't talk about the unique line going through  $L$  and  $M$ . We therefore define the line we use as the point tangent to the elliptic curve at point  $L$ . If  $L$  and  $M$  are

---

<sup>4</sup>The more precise reader may object to our usage of  $Z_p$  instead of  $GF(p)$  to denote the finite field with  $p$  elements. We abuse the notation to make more explicit the connections with later sections, which will deal with arithmetic over the ring of integers mod  $n$ . This structure we will denote by  $Z_n$ , again a slight abuse of terminology.



on a vertical line, the line clearly won't hit the curve at any other point. In this case, we define  $L + M$  to be equal to  $I$ . Even if  $L$  and  $M$  aren't on a vertical line, it still may be the case that the line through  $L$  and  $M$  doesn't intersect the curve at any other point. In this case, it can be shown that the line will be tangent to the curve at one of the two points of intersection. We treat this tangency as a double point of intersection, and use it as the "third" point.

For obvious reasons, this pictorial algorithm for adding two points is called the "tangent and chord" method. While intuitively appealing, we require a computational algorithm for adding points, and we also need to generalize the addition operation to arbitrary fields. Fortunately, the geometric operations required for the tangent and chord method can be expressed algebraically. The resulting algorithm is given in Figure 2.2. This algorithm works for arbitrary fields such that  $2, 3 \neq 0$ .

```

ALGORITHM ADD( $(x_1, y_1), (x_2, y_2), (A, B)$ )
if  $x_1 = x_2$  and  $y_1 = y_2$  then return( $I$ )
if  $x_1 = x_2$  then  $\lambda = \frac{3x_1^2 + A}{2y_1}$ 
    else  $\lambda = \frac{y_2 - y_1}{x_2 - x_1}$ 
 $\beta = y_1 - \lambda x_1$ 
 $x_s = \lambda^2 - x_1 - x_2$ 
 $y_s = -(\lambda x_s + \beta)$ 
return( $(x_s, y_s)$ )

```

Figure 2.2: Algorithm for adding two points on an elliptic curve.

### 2.3.3 The group structure of curves over $Z_p$

For our algorithm, we use some classical results about curves over  $Z_p$ , as well as some more recent results. First, the set of points of the elliptic curve  $(A, B)$  over  $Z_p$  form an abelian group, where the group addition operation is defined as above. This group is isomorphic to  $Z_{m_1} \times Z_{m_2}$  for some  $m_1, m_2$ , where  $m_1 | m_2$ . Here,  $Z_{m_i}$  denotes the cyclic additive group of integers mod  $m_i$ . For notational convenience, we will also use  $Z_n$  to denote the ring of integers mod  $n$ . We will explicitly state whenever we mean for  $Z_n$  to be viewed as a cyclic additive group.

For our purposes, we will need to know a great deal about the sizes of groups formed by elliptic curves over  $Z_p$ . For ease of notation, we make the following definition.

**Definition 2.8** Given an elliptic curve  $(A, B)$ , we denote by  $\#_p(A, B)$  the number of points on  $(A, B)$ .

A great deal is known about the sizes of groups over  $Z_p$ . First, there is the classical result, known as the *Riemann Hypothesis for Finite Fields*. This result implies that

$$p + 1 - 2\sqrt{p} \leq \#_p(A, B) \leq p + 1 + 2\sqrt{p},$$

for all curves  $(A, B)$  over  $Z_p$ , and all  $p \neq 2, 3$ .

This theorem is very useful to us, since we will need to bound the sizes of groups to use our methodology. However, in order to analyze our algorithms, it is also necessary to have some understanding of the distribution of

$\#_p(A, B)$  when  $(A, B)$  is uniformly distributed. The following theorem, due to Lenstra[L], proves crucial to our attack.

**Theorem 2.4** [Lenstra] Let  $p > 5$  be a prime. Let

$$S \subseteq [p + 1 - \lfloor \sqrt{p} \rfloor, p + 1 + \lfloor \sqrt{p} \rfloor].$$

If curve  $(A, B)$  over  $Z_p$  is chosen uniformly, then

$$\text{prob}(\#_p(A, B) \in S) > \frac{c}{\ln p} \cdot \frac{|S| - 2}{2\lfloor \sqrt{p} \rfloor + 1},$$

where  $c$  is some fixed constant.

This theorem relates the probability of finding randomly selecting a curve of a “good” order to the probability of randomly selecting a “good” integer, where “good” can be arbitrarily defined. If one uniformly picks a number from the interval  $[p + 1 - \lfloor \sqrt{p} \rfloor, p + 1 + \lfloor \sqrt{p} \rfloor]$  the probability that it will be in  $S$  is exactly

$$\frac{|S|}{2\lfloor \sqrt{p} \rfloor + 1}.$$

Essentially, the size of a random group is at most  $O(1/\ln p)$  times less likely to have a particular property as a randomly selected integer in

$$[p + 1 - \lfloor \sqrt{p} \rfloor, p + 1 + \lfloor \sqrt{p} \rfloor],$$

provided that  $|S| > 2$ .

Finally, we also use an algorithm due to Schoof[Sc] which, when given a curve  $(A, B)$  over  $Z_p$ , will deterministically compute  $\#_p(A, B)$  in  $O(|p|^c)$  steps. Schoof’s original implementation,  $c = 9$ . Lenstra has since improved the algorithm’s exponent to 8.

### 2.3.4 The structure of elliptic curves over $Z_n$ , for composite $n$ .

Given a value of  $n$ , we can consider elliptic curves over  $Z_n$ , and add points according to our algorithm. However, if  $n$  is composite, then the structure

we get will not be a group.<sup>5</sup> Indeed, the addition operation will not even be defined, since the inverse elements required by our algorithm may not exist.

While we lose a some algebraic structure by considering elliptic curves over  $Z_n$  for arbitrary  $n$ , some useful properties remain. We can treat these structures as weak structures which project down to elliptic curves over  $Z_p$ , for  $p|n$ . This will allow us use the abstract machinery of the last section to generate a new primality criterion.

### Formal definitions.

We formally define elliptic curves over  $Z_n$ , and the basic operations on the points of these curves, as follows.

**Definition 2.9** Let  $n$  be an integer not divisible by 2 or 3. An elliptic curve over  $Z_n$  is defined as an ordered pair  $(A, B)$ , where  $A, B \in Z_n$ , and  $(4A^3 + 27B^2, n) = 1$ . The points on  $(A, B)$  are defined to be the set of ordered pairs  $(x, y)$  such that  $y^2 = x^3 + Ax + B \pmod n$ , along with an additional element  $I_n$ . We denote this set of points by  $E_{A,B}(n)$ . The addition operation for two points  $L, M \in E_{A,B}(n)$  is defined by  $I_n + M = M + I_n = M$ , if applicable, and otherwise by result of algorithm  $\text{ADD}(L, M, (A, B))$ . If  $\text{ADD}(L, M, (A, B))$  ever needs to compute an inverse which is not defined over  $Z_n$ , then  $L + M$  is undefined.

We also define scalar multiplication of points on these curves. We define  $qL$ , for  $q > 0$  and  $L \in E_{A,B}(n)$ , as we defined exponentiation in Section 2.2.2, replacing multiplication with addition. Thus, we have

$$qL = \begin{cases} L & \text{for } q = 1, \\ (L + L)^{q/2} & \text{for } q \text{ even}, \\ L + (q - 1)L & \text{for } q \text{ odd}. \end{cases}$$

The reader can verify that this is a purely syntactic transformation from multiplicative to additive notation.

It should be pointed out that the definitions and operations described here are perfectly compatible with the definitions given for curves over  $Z_p$ . Thus, if  $n$  happens to be prime, everything we will say in this section will still hold.

---

<sup>5</sup>Lenstra has pointed out that there is a relatively natural addition operation for which the points over  $Z_n$  do form a group. However, we do not use it in this exposition.

### A projection from curves over $Z_n$ to curves over $Z_p$ .

Given an elliptic curve  $(A, B)$  over  $Z_n$ , and a prime  $p > 3$  that divides  $n$ , there is a natural projection from  $(A, B)$  to an elliptic curve over  $Z_p$ . First, we recall our notation for projections: Given an element  $x \in Z_n$ , we define  $x_p$  to be equal to the element in  $Z_p$  obtained by taking  $x \bmod p$ . We define our projection,

$$\tau_p^n : E_{A,B}(n) \rightarrow E_{A_p,B_p}(p),$$

as follows. First, we define  $\tau_p^n(I_n) = I_p$ , where  $I_p$  is the identity for  $E_{A_p,B_p}(p)$ . Otherwise, we define  $\tau_p^n((x, y)) = (x_p, y_p)$ .

To see that this projection makes sense, we use the fact that the mapping from  $x$  to  $x_p$  preserves both addition and multiplication. We first note that if  $4A_p^3 + 27B_p^2 = 0 \bmod p$ , then  $4A^3 + 27B^2 = 0 \bmod p$ . Since this would imply  $p \mid (4A^3 + 27B^2, n)$ , and  $(4A^3 + 27B^2, n) = 1$ , we have

$$4A_p^3 + 27B_p^2 \not\equiv 0 \bmod p.$$

Thus,  $(A_p, B_p)$  will indeed be an elliptic curve over  $Z_p$ . Second, if

$$y^2 = x^3 + Ax + B \bmod n, \text{ then,}$$

$$y_p^2 = x_p^3 + A_p x_p + B_p \bmod p.$$

Thus, the mapping  $\tau_p^n$  projects points of  $(A, B)$  to points of  $(A_p, B_p)$ .

In the following lemma, we show that this projection preserves addition of points.

**Lemma 2.3** Let  $n$  be an integer that is not divisible by 2 or 3, and let  $p$  be a prime that divides  $n$ . Let  $(A, B)$  be a curve over  $Z_n$ , and let  $L, M \in E_{A,B}(n)$ . Then if  $L + M$  is defined, we have

$$\tau_p^n(L + M) = \tau_p^n(L) + \tau_p^n(M).$$

**Proof:** The proof of this lemma follows straightforwardly from a simple case analysis on the rules for adding two points on an elliptic curve. These rules may be summarized as:

1.  $I + M = M + I = M$ ,
2.  $(x, y) + (x, -y) = I$ ,

3. For  $y \neq 0$ ,

$$(x, y) + (x, y) = \left( \frac{P_1(x, y, A, B)}{2y}, \frac{P_2(x, y, A, B)}{2y} \right),$$

4. For  $x_1 \neq x_2$ ,

$$(x_1, y_2) + (x_2, y_2) = \left( \frac{Q_1(x_1, y_1, x_1, y_2, A, B)}{x_1 - x_2}, \frac{Q_2(x_1, y_1, x_1, y_2, A, B)}{x_1 - x_2} \right)$$

Here,  $P_1, P_2, Q_1, Q_2$  are fixed polynomials. For our proof, we do not need any further information about these polynomials. Rather, we use the following elementary fact from number theory: Let  $R(a, b, c, \dots)$  be a rational function<sup>6</sup>. If  $x = R(a, b, c, \dots)$  is well-defined for a particular assignment of  $a, b, c, \dots$  over  $Z_n$ , then  $R(a_p, b_p, c_p, \dots)$  will be well-defined over  $Z_p$ , and will be equal to  $x_p$ . In other words, the trivial projection from  $Z_n$  to  $Z_p$  preserves addition, subtraction, multiplication, and divisions that are well-defined (i.e. do not require inverses which do not exist) over  $Z_n$ .

We now go through the above four cases, and show that the lemma holds for each of them.

In the first case, we note that if  $L$  or  $M$  is equal to  $I_n$  (the identity of  $E_{A,B}(n)$ ), the lemma holds trivially. For the rest of the analysis, we can thus assume that  $L, M \neq I_n$ , and hence that  $\tau_p^n(L), \tau_p^n(M) \neq I_p$  (Here,  $I_p$  is the identity of  $E_{A_p, B_p}(p)$ ).

In the second case, we note that if  $a + b \equiv 0 \pmod n$  then  $a_p + b_p \equiv 0 \pmod p$ . Therefore, if  $L, M$  are of the form

$$L = (x, y), M = (x, -y),$$

then  $\tau_p^n(L), \tau_p^n(M)$  will be of the form

$$\tau_p^n(L) = (x_p, y_p), \tau_p^n(M) = (x_p, -y_p).$$

Thus,  $L + M = I_n$  and  $\tau_p^n(L) + \tau_p^n(M) = I_p$ , so the lemma holds.

In the third case, we have  $L = M = (x, y)$ , where  $y \neq 0$ . By the definition of  $\tau_p^n$ , we have  $\tau_p^n(L) = \tau_p^n(M) = (x_p, y_p)$ . Thus, when we add  $\tau_p^n(L) + \tau_p^n(M)$ ,

---

<sup>6</sup>i.e. a ratio of two polynomials

we use Rule 2 when  $y_p = 0$ , and Rule 3 otherwise. If  $y_p = 0$ , we have  $p|y$ , which implies that  $(2y)^{-1}$  does not exist in  $Z_n$ . However, Rule 3 implies that  $L + M$  is defined iff  $(2y)^{-1}$  exists in  $Z_n$ . Therefore,  $\tau_p^n(L)$  and  $\tau_p^n(M)$  must also be added using Rule 3. Thus, we have,

$$\begin{aligned} L + M &= \left( \frac{P_1(x, y, A, B)}{2y}, \frac{P_2(x, y, A, B)}{2y} \right), \text{ and,} \\ \tau_p^n(L) + \tau_p^n(M) &= \left( \frac{P_1(x_p, y_p, A_p, B_p)}{2y_p}, \frac{P_2(x_p, y_p, A_p, B_p)}{2y_p} \right). \end{aligned}$$

Using the result about projections of rational functions, we therefore have  $\tau_p^n(L + M) = \tau_p^n(L) + \tau_p^n(M)$ .

In the fourth case, we have  $L = (x_1, y_1)$  and  $M = (x_2, y_2)$ , where  $x_1 \neq x_2$ . This case is virtually identical to the third case. By the definition of  $\tau_p^n$ , we have  $\tau_p^n(L) = (x_{1p}, y_{1p})$  and  $\tau_p^n(M) = (x_{2p}, y_{2p})$ . If

$$x_{1p} \equiv x_{2p},$$

then  $p|x_1 - x_2$ , and thus  $(x_1 - x_2)^{-1}$  will have no inverse mod  $n$ . However, by Rule 4,  $L + M$  will be defined iff  $(x_1 - x_2)^{-1}$  exists, so

$$x_{1p} \not\equiv x_{2p} \pmod{p}.$$

Therefore,  $\tau_p^n(L)$  and  $\tau_p^n(M)$  are also added using Rule 4. Hence, we have,

$$\begin{aligned} L + M &= \left( \frac{Q_1(x_1, y_1, x_1, y_2, A, B)}{x_1 - x_2}, \frac{Q_2(x_1, y_1, x_1, y_2, A, B)}{x_1 - x_2} \right), \text{ and,} \\ \tau_p^n(L) + \tau_p^n(M) &= \left( \frac{Q_1(x_{1p}, y_{1p}, x_{1p}, y_{2p}, A_p, B_p)}{x_{1p} - x_{2p}}, \frac{Q_2(x_{1p}, y_{1p}, x_{1p}, y_{2p}, A_p, B_p)}{x_{1p} - x_{2p}} \right). \end{aligned}$$

Again using the result about projections of rational functions, we have  $\tau_p^n(L + M) = \tau_p^n(L) + \tau_p^n(M)$ . ■

### 2.3.5 A primality criterion using elliptic curves.

Lemma 2.3 is the key result we need to apply our general methodology to elliptic curves. We give resulting primality criterion below.

**Theorem 2.5** Let  $n$  be an integer, not divisible by 2 or 3. Let  $(A, B)$  be an elliptic curve over  $Z_n$ , and let  $L \in E_{A,B}(n)$ , with  $L \neq I_n$ . Suppose that  $qL = I_n$ , for some  $q > n^{1/2} + 2n^{1/4} + 1$ . Then if  $q$  is prime,  $n$  will be prime.

**Proof:** We make a direct appeal to our general primality criterion. Let our group assignment  $\gamma$  be defined as follows. First, if  $p = 2$  or  $3$ , then we define  $\gamma(p) = \{\}$ . Otherwise, we define  $\gamma(p)$  to be equal to the set of groups generated by elliptic curves over  $Z_p$ . Since for any elliptic curve  $(Q, R)$  over  $Z_p$ ,

$$\#_p(Q, R) \leq p + 2\sqrt{p} + 1,$$

we have

$$S(n, \gamma) \leq n^{1/2} + 2n^{1/4} + 1.$$

Thus,  $q > S(n, \gamma)$ .

By Lemma 2.3,  $E_{A,B}(n)$  will  $(n, L)$ -project for  $n$  not divisible by 2 or 3, and  $L \neq I$ . Note that by definition,  $\tau_p^n(L) \neq I_p$  if  $L \neq I_n$ . Hence, by Theorem 2.3, if  $q$  is prime,  $n$  will be prime. ■

## 2.4 Our primality proving algorithm.

Given the primality criterion developed in the last section, it is now a fairly simple matter to create a primality proving algorithm. On the highest level, we will use our primality criterion to reduce the primality of a prime  $p$  to the primality of a prime  $q$ , where  $q = p/2 + o(p)$ . We then recursively prove that  $q$  is prime. For technical reasons, we eventually stop when the number to be proven prime is sufficiently small that it may be deterministically verified as prime. If too much time passes, the algorithm times out, and starts from scratch.

In this section, we first outline our main reduction step, i.e. how we reduce the primality of  $p$  to the primality of  $q$ . Then we show how this step is used in the complete algorithm.

### 2.4.1 Implementing the main step of the algorithm.

The main step of our algorithm is as follows: Given a prime  $p$ , we will construct a curve over  $Z_p$ , and a point on this curve with a large prime order,  $q$ . This will reduce the problem of proving  $p$  prime to the problem of

proving  $q$  prime. The value of  $q$  will be substantially less than the value of  $p$ , so quantifiable progress will be made by such a reduction.

We find a curve along with a point of high prime order in two randomized stages. First, we find a curve  $(A, B)$  that we know has a large number of points of prime order  $q$ . Then we randomly pick points  $L \in E_{A,B}(p)$  until we find one such that  $L \neq I$  and  $qL = I$ .

### Picking a good curve.

The first step of this procedure is accomplished using Schoof's algorithm for computing the number of points on an elliptic curve. We first uniformly choose  $A, B$  such that  $(4A^3 + 27B^2, p) = 1$ . We do this by uniformly choosing a pair  $(A, B)$ , and throwing it out if  $(4A^3 + 27B^2, p) > 1$ . For prime  $p$ , and for any value of  $A$ , there are at most two values of  $B$ ,  $\pm\sqrt{-4A^3/27}$  (computed over  $Z_p$ ) which will yield a bad pair. Thus, with overwhelming probability, a randomly chosen  $(A, B)$  will constitute an elliptic curve. Once we have a curve,  $(A, B)$ , we compute its order using Schoof's algorithm. If  $\#_p(A, B) = 2q$ , where  $q$  is a prime, then we stop. Otherwise, we pick a new pair  $(A, B)$ , and start the process over again.

The alert reader will ask, "How do you tell if  $q$  is prime?" We in fact use a probabilistic primality test, run  $2|p|$  times so that the probability of mistaking a composite  $q$  as prime is less than  $1/p$ . This introduces a small probability of error into our algorithm. However, as we will see later, it will be possible to correct such errors before they can cause an incorrect output.

We give the curve generation algorithm in Figure 2.3.

We now compute the expected running time of this step. Our procedure for finding a curve  $(A, B)$  of order  $2q$  will take expected time equal to the expected time necessary to generate and test a single curve, multiplied by the expected number of curves it must try. The time necessary to test a curve is dominated by Schoof's algorithm, which takes  $O(|p|^8)$  steps. The expected time necessary to generate a curve, and to run the probabilistic primality tests are lower order polynomials in  $|p|$ .

We cannot give such a nice, closed form formula for the number of curves we must test before coming up with one whose order is twice a prime. However, we can use Lenstra's theorem to relate this number to the size of the

**Algorithm Generate-Curve( $p$ )**

- 1: Uniformly generate  $(A, B)$  until  $(4A^3 + 27B^2, p) = 1$ .
- 2: Compute  $\#_p(A, B)$  using Schoof's algorithm. If this number is odd, go to Step 1. Otherwise, set  $q = \#_p(A, B)/2$ .
- 3: Run the probabilistic test of [SS] on  $q$  for  $2|p|$  trials. If one of the trials ever outputs "composite", go to Step 1. If  $2, 3|q$ , go to Step 1.
- 4: Return $((A, B), q)$

Figure 2.3: Algorithm for generating a curve of order  $2q$ , where  $q$  is prime.

set  $S(p)$  defined by

$$S(p) = \left\{ q \in \left[ \frac{p+1 - \lfloor \sqrt{p} \rfloor}{2}, \frac{p+1 + \lfloor \sqrt{p} \rfloor}{2} \right], q \text{ prime.} \right\}.$$

**Lemma 2.4** Let  $p > 5$  be a prime, and let  $(A, B)$  be chosen uniformly from curves over  $Z_p$ . Let  $S(p)$  be defined as above. Then

$$\text{prob}(\#_p(A, B) \text{ is twice a prime}) > \frac{c}{\lg p} \cdot \frac{|S(p)| - 2}{2\lfloor \sqrt{p} \rfloor + 1},$$

where  $c$  is some fixed constant.

**Proof:** There is a trivial bijection between numbers in the interval

$$[p+1 - \lfloor \sqrt{p} \rfloor, p+1 + \lfloor \sqrt{p} \rfloor]$$

which are twice a prime, and elements of  $S(p)$ . Applying Lenstra's theorem immediately gives the desired bound. ■

We will not be interested in the lower-order terms of the above formula, which only come into play when  $S(p)$  is far too small to be of any use. We therefore state the following simple corollary to Lemma. 2.4, which follows by a simple calculation.

**Corollary 2.5** Let  $p > 5$  be prime, and that  $S(p) = O(\sqrt{p}/\lg^c p)$ . Then algorithm GENERATE-CURVE( $p$ ) will select at most expected  $O(|p|^{c+1})$  curves, running for expected  $O(|p|^{c+9})$  steps before it terminates. ■

### Picking a point on a curve.

Once we have a curve  $(A, B)$  of order  $2q$ , where  $q$  is prime, it is a simple matter to find a point of order  $q$  on  $(A, B)$ . Recall that group  $E_{A,B}(p)$  is isomorphic to a product of cyclic additive groups,  $Z_{m_1} \times Z_{m_2}$ , where  $m_1 | m_2$ . Since  $E_{A,B}(p)$  is of size  $2q$ , we have  $m_1 m_2 = 2q$ , and hence  $m_1 = 1, m_2 = 2q$ , for  $q > 2$ . Thus  $E_{A,B}(p)$  will in fact be isomorphic to  $Z_{2q}$ . It is a simple exercise to verify that  $Z_{2q}$  has  $q - 1$  points of order  $q$ , and thus, by our isomorphism, so must  $E_{A,B}(p)$ .

Since almost half the points of  $E_{A,B}(p)$  will have the desired property, we merely have to show how to efficiently generate points  $L$  at random on  $(A, B)$ , and how to efficiently determine if  $qL = I$ .

To generate a point on  $(A, B)$ , we simply generate a point  $x \in Z_p$ , and check if  $x^3 + Ax + B$  is a quadratic residue. This naively takes  $O(|p|^3)$  time. If not, we repeat the process. Otherwise, we compute  $y = \sqrt{x^3 + Ax + B}$ , using the algorithm of Adleman, Manders, and Miller [AMM] to take square roots modulo a prime, and randomly choose which of the square roots to take. Clearly, the curve can have at most 2 points with a given  $x$  coordinate, and hence there must be  $O(p)$  different  $x$  coordinates such that  $x^3 + Ax + B$  is a quadratic residue. Thus, we will have to test an expected  $O(1)$  values of  $x$  before being able to pick the value of  $y$ . Taking square roots takes  $O(|p|^3)$  expected time, so the total algorithm naively takes  $O(|p|^3)$  time. This time, is negligible compared to the  $O(|p|^{9+c})$  time required by the first part of the main step (which is why we were not concerned about getting the smallest exponent possible).

We have cheated here, but only infinitesimally. A point of the form  $(x, 0)$  will be chosen twice as often as any other point, since both square roots of 0 are the same. Also, the identity is never chosen by this method, which is actually what we want. This deviation from the uniform distribution is exponentially small, and thus insignificant. If the earth were to explode whenever a point of the form  $(x, 0)$  were chosen, we would hardly ever notice it.

**Algorithm Select-Point( $p, q, (A, B)$ )**

- 1: Choose  $x$  uniformly from  $Z_p$ , and set  $z = x^3 + Ax + B$ .
- 2: Test if  $z$  is a quadratic residue. If it is not, then go to Step 1.
- 3: Compute  $y = \sqrt{z}$ , uniformly choosing which square root of  $z$  to take. Set  $L = (x, y)$ .
- 4: Compute  $qL$ . If  $qL \neq I_p$ , then go to Step 1. Otherwise, return( $L$ ).

Figure 2.4: Algorithm for choosing a point on  $(A, B)$  of order  $q$ .**Algorithm Main-Step( $p$ )**

- 1: Make the following computations:

$$(A, B), q \leftarrow \text{GENERATE-CURVE}(p),$$

$$L \leftarrow \text{SELECT-POINT}(p, q, (A, B))$$

Return( $(A, B), L, q$ ).

Figure 2.5: Main reduction step of the primality-proving algorithm.

Checking if  $qL = I$  is straightforward, given our implicit algorithm for point addition, and the “repeated doubling” trick for exponentiation, discussed in the previous two sections. It naively requires  $O(|p|^2)$  steps to add two points, and  $O(\lg q) = O(|p|)$  total additions to multiply a point by  $q$ . Thus, at most  $O(|p|^3)$  steps are naively required. This bound can certainly be improved, but is already negligible compared to the first part of the main step.

For future reference, we give the our point selection algorithm in Figure 2.4, and the full algorithm for the main step in Figure 2.5.

### 2.4.2 Incorporating the main reduction step in the primality proving algorithm.

Given the main step shown in the previous section, we now can give the full primality-proving algorithm. The full algorithm essentially iterates the main reduction algorithm until the prime to be proven is so small that it may be verified prime in time polynomial in  $|p|$ , the size of the original prime number to be certified. We also have an abort condition, so that the algorithm will restart after sufficiently many steps have taken place. This mechanism is to handle exponentially rare circumstances, caused by the use of a probabilistic primality test, in which the algorithm will get caught in an infinite loop. We give the complete algorithm in Figure 2.6. For the exposition of this algorithm, we let constant  $C$  to be defined as a positive constant such that the Cohen-Lenstra primality test takes  $O(|p|)$  time on inputs of size,

$$2^{(\lg p)^{C/\lg \lg p}}.$$

It is easily verified that such a positive constant does exist.

We now show that the output of  $\text{PROVE-PRIME}(p)$  constitutes a certificate of  $p$ 's primality, that can be deterministically checked in time  $O(|p|^4)$ . Our deterministic checker works as follows. On input

$$p, ((A_0, B_0), L_0, p_1), \dots, ((A_{i-1}, B_{i-1}), L_{i-1}, p_i),$$

the algorithm first checks that  $p_i$  is small enough to be rapidly verified prime, and aborts if this is not the case. It then verifies that  $p_i$  is prime, and aborts if it is not the case. Then, for  $j = 1, \dots, i - 1$ , it verifies that

- $p_j$  is not divisible by 2 or 3,
- $(A_j, B_j)$  is a curve over  $Z_{p_j}$ ,
- $p_{j+1} > p_j^{1/2} + 2p_j^{1/4} + 1$ , and
- $L_j \neq I_{p_j}, q_j L_j = I_{p_j}$ .

We give this algorithm in Figure 2.7.

The following theorem shows that the output of our primality prover is indeed a certificate of primality.

**Algorithm Prove-Prime( $p$ )**

1: Let

$$lowerbound = \max \left( 2^{(\lg p)^{c/(\lg \lg p)}}, 37 \right),$$

where  $c$  is a suitably small constant. Let  $i = 0$ , and  $p_0 = p$ .

2: While  $p_i > lowerbound$ , do

$$(A_i, B_i), L_i, p_{i+1} \leftarrow \text{MAIN-STEP}(p_i),$$

and set  $i = i + 1$ . For each  $p_i$ , check if it is divisible by 2 or 3, and go to Step 1 if this is the case.

3: Using the deterministic algorithm of Cohen-Lenstra, check if  $p_i$  is prime. If  $p_i$  is not prime, then go to Step 1. Otherwise,

$$\text{Output}((A_0, B_0), L_0, p_1), \dots, ((A_{i-1}, B_{i-1}), L_{i-1}, p_i)$$

\*: If, since starting Step 1, more than  $|p|^{\lg |p|}$  steps have been run, abort and go to Step 1.

Figure 2.6: Why Joe Kilian should get his doctorate.

**Algorithm Check**  $(p, ((A_0, B_0), L_0, p_1), \dots, ((A_{i-1}, B_{i-1}), L_{i-1}, p_i))$

1: If

$$p_i > \max \left( 2^{(\lg p)^{C/\lg \lg p}}, 37 \right),$$

then abort. Otherwise, test  $p_i$  for primality, using the Cohen-Lenstra algorithm.

2: Define  $p_0 = p$ . For  $j \in [0, i-1]$ , check that

- $p_i$  is not divisible by 2 or 3,
- $(4A_j^3 + 27B_j^2, p_j) = 1$ ,
- $p_{j+1} > p_j^{1/2} + 2p_j^{1/4} + 1$ , and
- $L_j \neq I_{p_j}, p_{j+1}L_j = I_{p_j}$ .

If any of these conditions do not hold, abort. Otherwise, accept  $p$  as prime.

Figure 2.7: Algorithm for checking certificates of primality.

**Theorem 2.6** If algorithm  $\text{CHECK}(p, \text{certificate})$  accepts, then  $p$  is prime. For a prime  $p$ , let certificate be an output of algorithm  $\text{PRIME-PROVE}(p)$ . Then  $\text{CHECK}(p, \text{certificate})$  will accept in  $O(|p|^4)$  deterministic time.

Note that this theorem makes no guarantee as to how quickly, if ever,  $\text{PRIME-PROVER}$  will output a certificate for  $p$ , merely that such a certificate will be valid.

**Proof:** Suppose that  $\text{CHECK}$  accepts an input of the form

$$p, ((A_0, B_0), L_0, p_1), \dots, ((A_{i-1}, B_{i-1}), L_{i-1}, p_i).$$

Then clearly,  $p_i$  must be prime. Furthermore, by Theorem 2.5, the checks made for each value of  $j$  ensures that if  $p_{j+1}$  prime, then  $p_j$  is prime. Thus, if  $\text{CHECK}$  accepts, then

$$p_i \text{ prime} \implies p_{i-1} \text{ prime} \implies \dots \implies p_0 = p \text{ prime}.$$

Thus,  $p$  must be prime.

We now show that  $\text{CHECK}$  will always accept a certificate, of the above form, presented to it by  $\text{PRIME-PROVER}$ . We first note that by the definition of  $\text{PROVE-PRIME}$ ,  $p_i$  will be prime. By the definition of  $\text{GENERATE-CURVE}$ , we have  $(4A_j^3 + 27B_j^2, p_j) = 1$ . From the definition of  $\text{GENERATE-CURVE}$ , and the fact that  $\#_{p_j}(A_j, B_j) \geq p_j + 1 - 2\sqrt{p_j}$ , we have

$$\begin{aligned} p_{j+1} &\geq p_j + 1 - 2\sqrt{p_j} \\ &> p_j^{1/2} + 2p_j^{1/4} + 1, \end{aligned}$$

for  $p_j > 37$ . By the definition of  $\text{PROVE-PRIME}$ ,  $p_j > 37$ , unless  $p \leq 37$ , in which case it is easily verified that the output of  $\text{PROVE-PRIME}$  will be accepted by  $\text{CHECK}$ . Finally, by the definition of  $\text{SELECT-POINT}$ ,  $L_j \neq I_{p_j}$ , and  $p_{j+1}L_j = I_{p_j}$ . Therefore, check will accept.

To compute how many steps are required, we first note that  $p_{j+1} = p_j/2 + o(p_j)$ , and therefore  $i = O(\lg p)$ . For each value of  $j$  the checking procedure must perform a constant number of simple arithmetic operations, a single GCD computation, and must multiply a point  $L_j$  by an integer  $q_j$ . This all can be done in  $O(|p|^3)$ , so the total running time of the checking algorithm is  $O(|p|^3) \cdot O(\lg p) = O(|p|^4)$  steps. ■

## 2.5 Analysis of the primality proving algorithm.

In the previous section, we exhibited our primality proving algorithm, and demonstrated that it produced legitimate certificates of primality. We also gave the running-time analysis of the main step of the algorithm, as a function of the number of primes in certain intervals.

In this section, we analyze how long it takes for this algorithm to produce proofs of primality. We show that, modulo a conjecture on the distribution of prime numbers, the algorithm will always halt in expected polynomial time. We then extend this argument to show that the algorithm will produce proofs of primality, in expected polynomial time, for all but a vanishing fraction of the prime numbers. This latter theorem does not depend on any conjectures.

### 2.5.1 Analysis based on a conjecture.

Using the machinery of the previous sections, it is straightforward to analyze the running time of our algorithm under the assumption that all the small intervals we need consider have sufficiently many primes.

**Theorem 2.7** Suppose that

$$(\exists c_1, c_2 > 0) \pi(x + \sqrt{x}) - \pi(x) \geq \frac{c_2 \sqrt{x}}{\log^{c_1} x}.$$

then algorithm `PROVE-PRIME`( $p$ ) will terminate in expected time  $O(|p|^{c_1+10})$ .

**Proof:** It is not hard to see that the bulk of steps of `PROVE-PRIME` occur within algorithm `MAIN-STEP`. Since in our primality-proving algorithm,  $p_{j+1} = p_j/2 + o(p_j)$ , only  $O(\ln p)$  iterations of `MAIN-STEP` are invoked. The expected time taken by algorithm `MAIN-STEP` is equal to the expected time taken by algorithm `GENERATE-CURVE`, plus the expected time of algorithm `SELECT-POINT`. Algorithm `SELECT-POINT` is easily seen to take  $o(|p|^{10})$  time, provided that in the output,  $((A, B), q)$  of `GENERATE-CURVE`,  $q$  is indeed prime. We therefore have only to bound the total expected running time of the  $O(\ln p)$  iterations of `GENERATE-CURVE`.

First, let us assume for simplicity that each input  $p_j$  to `GENERATE-CURVE` is prime, and that the time-out procedure is never invoked. Recall that  $S(p_j)$

is defined by

$$S(p_j) = \left\{ q \in \left[ \frac{p_j + 1 - \lfloor \sqrt{p_j} \rfloor}{2}, \frac{p_j + 1 + \lfloor \sqrt{p_j} \rfloor}{2} \right], q \text{ prime.} \right\}.$$

We first simplify this expression. Setting  $x = (p_j + 1 - \lfloor \sqrt{p_j} \rfloor)/2$ , and  $y = (p_j + 1 + \lfloor \sqrt{p_j} \rfloor)/2$ , we have have

$$S(p_j) = \{q \in [x, y], q \text{ prime.}\}.$$

For  $p > 37$  (37 is the smallest value of  $p_j$  we need consider), we can easily verify that  $y > x + \sqrt{x}$ . Thus, there must be  $O(\frac{\sqrt{x}}{\log^{c_1} x})$  primes in  $S(p_j)$ . Therefore, by Corollary 2.5, `GENERATE-CURVE`( $p_j$ ) will take expected  $O(|p_j|^{c_1+9})$  steps. Thus, the total running time of the algorithm will, in this optimistic scenario, take  $O(|p|^{c_1+10})$  steps.

In actuality, we cannot always guarantee that the numbers which are supposed to be prime will actually be prime. This is because the probabilistic primality test we used could conceivably label a composite number as prime. In this case, we cannot bound the amount of time any of the subroutines will take. Thus, whenever a primality test is invoked, we have the possibility of going into an infinite loop should the test make a mistake.

We now show that mistakes in probabilistic primality testing will not be frequent enough or costly enough to increase the expected running time of the algorithm. We have set the probability of the probabilistic primality test failing to be at most

$$\frac{1}{p_j} \leq 2^{-(\ln p)^{C/\ln \ln p}}.$$

Our time-out feature causes us to rerun the algorithm whenever  $|p|^{\ln |p|}$  steps have elapsed. Thus, all a mistake could cost us in the worst case is to spend  $|p|^{\ln |p|}$  steps, and have to start over from scratch. First, we observe that the penalty caused by spending  $|p|^{\ln |p|}$  steps is unimportant. Whenever we have a primality test, we have an expected penalty of at most,

$$|p|^{\ln |p|} \cdot 2^{-(\ln p)^{C/\ln \ln p}} = o(1),$$

steps run due to possible infinite loops, which is negligible. We may therefore assume, without affecting the analysis, that as soon as a primality testing mistake is made, the algorithm simply starts over from scratch.

We now need only to bound the cost of having to start over whenever a primality test makes a mistake, or whenever the algorithm times-out after  $|p|^{\ln |p|}$  steps have occurred. By the above analysis, we have at least a nonzero constant probability that no more than  $O(|p|^{c_1+10})$  steps will be run before the completion of the algorithm, provided that no primality-testing mistakes are made. In this case, the time-out feature will not be invoked, and, since at most  $O(|p|^{c_1+10})$  primality tests could conceivably be run (this is a terrible overestimate, but sufficient for our purposes), there will be at most a

$$O(|p|^{c_1+10}) \cdot 2^{-(\ln p)^{C/\ln \ln p}} = o(1)$$

chance that of a testing mistake occurring. Therefore, even in the nonideal scenario, there will be a nonzero constant probability that the algorithm goes from start (or restart) to completion, without being restarted by primality testing mistakes or time-outs. Thus, the expected running time ( $ERT(p)$ ) of the algorithm will obey the recurrence,

$$ERT(p) = c \cdot O(|p|^{c_1+10}) + (1 - c)ERT(p),$$

which clearly implies that  $ERT(p) = O(|p|^{c_1+10})$ . ■

We again note that the only danger from a primality test failing is in the running-time of the algorithm, not the correctness of its output.

## 2.5.2 Proving our algorithm fast for most primes.

The scenario in the previous section is optimistic. It assumes that whenever one is attempting to show a number  $p$  prime, there will always be sufficiently many primes in the interval

$$\left[ \frac{p+1 - \lfloor \sqrt{p} \rfloor}{2}, \frac{p+1 + \lfloor \sqrt{p} \rfloor}{2} \right].$$

That is,  $S(p)$  is assumed to be sufficiently large. This is almost certainly the case for all primes, but it is currently beyond our ability to prove this fact. However, it has been shown that intervals which contain a sparse number of primes are rare. We have the following result, which is implied by a technical lemma of Heath-Brown[HB].<sup>7</sup>

---

<sup>7</sup>Our deepest thanks to Maier and Pomerance for pointing out this implication of Heath-Brown's work.

**Theorem 2.8** [Heath-Brown] Call an integer  $y$  *sparse* if there are less than  $\sqrt{y}/2 \lfloor \ln y \rfloor$  primes in the interval  $[y, y + \lfloor \sqrt{y} \rfloor]$ . Then there exist a constant  $\alpha$  such that for sufficiently large  $x$ ,

$$|\{y : y \in [x, 2x], y \text{ is sparse}\}| < x^{5/6} \ln^\alpha x.$$

We can use this result to show that our algorithm is fast for most prime numbers. Specifically, we prove the following theorem:

**Theorem 2.9** Let  $k$  be sufficiently large. There exist constants  $c_1, c_2$  such that the number of  $k$ -bit primes that algorithm `PROVE-PRIME`( $p$ ) will not halt in expected  $c_1 k^{11}$  steps is at most

$$2^k / 2^{-k^{c_2 / \lg \lg k}}.$$

**Proof:** Given a prime  $p$ , we denote by  $P_i(p)$  the set of all intermediate primes that can be generated in step  $i$  of the algorithm. In other words,  $P_i(p)$  consists of all primes that could conceivably be equal to  $p_i$  in the certificate generated for  $p$ . Thus, for instance,

$$P_0(p) = \{p\}, P_1(p) \subseteq \left[ \frac{p+1 - \lfloor 2\sqrt{p} \rfloor}{2}, \frac{p+1 + \lfloor 2\sqrt{p} \rfloor}{2} \right], \dots$$

These are the only primes that need be considered for proving  $p$  prime. If it is the case that  $S(p_i)$  is  $O(\sqrt{p_i}/\ln p_i)$  for  $p_i \in P_i(p)$ , then by the same analysis as in the proof of Theorem 2.7, `PROVE-PRIME`( $p$ ) will terminate in expected time  $O(k^{11})$ . The rest of this proof consists of showing that this will be true for most primes.

Our proof proceeds in three stages. First, we establish a simple property about how spread out  $P_i(p)$  is, and use this property to derive a simple criterion which implies that  $S(p_i)$  is large for  $p_i \in P_i(p)$ . Next, we use a result of Heath-Brown, and a simple combinatoric argument, to show that our criterion will fail for only a relatively small number of values of  $P_i(p)$ . Finally, we use this result to bound the number of primes for which our algorithm is slow.

**Characterizing  $P_i(p)$ .**

We note that for every certificate,  $p_{i+1} = p_i/2 + o(p_i)$ . This would suggest  $P_i(p)$  should be clustered around  $p/2^i$ . We now show that such is the case.

**Lemma 2.6** Let  $p$  be a prime, and let  $p/2^i$  be sufficiently large. Then any element of  $P_i(p)$  lies in the range

$$(\frac{p}{2^i} - 7\sqrt{\frac{p}{2^i}}, \frac{p}{2^i} + 7\sqrt{\frac{p}{2^i}}).$$

**Remark:** The value of 7 which we obtain can be improved on. However, we only need to establish that some constant exists.

**Proof:** Our proof is by induction on  $i$ . For  $i = 0$ , the lemma clearly holds. We can bound the largest and smallest elements of  $P_i(p)$  in terms of the largest and smallest elements of  $P_{i-1}(p)$ . Specifically, we have

$$\begin{aligned} \max(P_i(p)) &\leq \frac{\max(P_{i-1}(p)) + 1 + 2\sqrt{\max(P_{i-1}(p))}}{2}, \text{ and,} \\ \min(P_i(p)) &\geq \frac{\min(P_{i-1}(p)) + 1 - 2\sqrt{\min(P_{i-1}(p))}}{2}. \end{aligned}$$

By inductive hypothesis, we have

$$\max(P_i(p)) \leq \frac{\frac{p}{2^{i-1}} + 7\sqrt{\frac{p}{2^{i-1}}} + 1 + 2\sqrt{\frac{p}{2^{i-1}} + 7\sqrt{\frac{p}{2^{i-1}}}}}{2}.$$

We can simplify the above expression considerably. We note that

$$x + 7\sqrt{x} < (1 + o(1))x,$$

for  $x$  sufficiently large,

$$(p/2^{i-1})/2 = p/2^i,$$

and

$$\sqrt{p/2^{i-1}} = \sqrt{2} \cdot \sqrt{p/2^i}.$$

These simplifications yield,

$$\begin{aligned}\max(P_i(p)) &< \frac{p}{2^i} + \left(\frac{7}{\sqrt{2}} + \sqrt{2} + o(1)\right)\sqrt{\frac{p}{2^i}} + 1 \\ &< \frac{p}{2^i} + 7\sqrt{\frac{p}{2^i}},\end{aligned}$$

For  $p/2^i$  sufficiently large. The lower bound is similarly established. ■

**A condition under which  $S(p_i)$  will be large.**

We can use Lemma 2.6 to give a simple condition under which we can guarantee that  $S(p_i)$  will be large for all  $p_i \in P_i(p)$ . To facilitate the discussion, we introduce the following terminology.

**Definition 2.10** Let  $\mathcal{I}_i(p)$  denote the set of intervals of the form

$$\left[ \frac{p_i + 1 - \lfloor \sqrt{p_i} \rfloor}{2}, \frac{p_i + 1 + \lfloor \sqrt{p_i} \rfloor}{2} \right],$$

where  $p_i \in P_i(p)$ .

In other words,  $\mathcal{I}_i(p)$  consists of the set of intervals which are important in our primality proving algorithm's search for  $p_{i+1}$ . If we can show that every interval in  $\mathcal{I}_i(p)$  has a large number of primes, then we are guaranteed that our algorithm will always be able to quickly generate  $p_{i+1}$ .<sup>8</sup> We would like to reduce statements about all the intervals in  $\mathcal{I}_i(p)$  to a statement about a relatively small number of intervals, for reasons which will become clear as our discussion progresses. This motivates the following definition.

**Definition 2.11** Let  $S_1$  be a set of intervals. We say that a set of intervals  $S_2$  is an *inclusion set* for  $S_1$  if

$$(\forall [x, y] \in S_1)(\exists [x', y'] \in S_2) x \leq x' \leq y' \leq y.$$

---

<sup>8</sup>The alert reader may ask why the  $\sqrt{p}$  term is not  $2\sqrt{p}$  in the definition of  $\mathcal{I}_i(p)$ . This is due to our definition of  $\mathcal{S}(p_i)$ , and indirectly due to fact that Lenstra's result only holds for the smaller interval.

In other words,  $S_2$  is an inclusion set for  $S_1$  if every interval in  $S_1$  has a subinterval that appears in  $S_2$ . The utility of this definition comes from the following simple observation: If every interval in  $S_2$  has at least  $n$  primes, then every interval in  $S_1$  will have at least  $n$  primes. We now show that  $\mathcal{I}_i(p)$  has a small inclusion set which consists of large intervals.

**Definition 2.12** Let  $\mathcal{C}_i(p)$  be defined as the set of intervals

$$\left\{ [x, x + \lfloor \sqrt{x} \rfloor] : x = \left\lfloor \frac{p}{2^{i+1}} + \frac{k}{3} \cdot \sqrt{\frac{p}{2^{i+1}}} \right\rfloor, k \in [-22, 22] \right\}.$$

**Remark:** For our application, the value of 22 can probably be reduced, but we only need the fact that some constant exists.

**Lemma 2.7** Let  $p$  be a prime, and let  $p/2^i$  be sufficiently large. Then  $\mathcal{C}_i(p)$  is an inclusion set for  $\mathcal{I}_i(p)$ .

**Proof:** First, let us characterize intervals  $[x, y] \in \mathcal{I}_i(p)$ . By the definition of  $\mathcal{I}_i(p)$ , and elementary algebra, we have,

$$y = x + (\sqrt{2} + o(1)) \sqrt{x}.$$

By the same argument as in Lemma 2.6, we have,

$$\frac{p}{2^{i+1}} + 7\sqrt{\frac{p}{2^{i+1}}} \geq x \geq \frac{p}{2^{i+1}} - 7\sqrt{\frac{p}{2^{i+1}}}.$$

Thus, there exist  $x_1, x_2$  such that  $x_1 \leq x \leq x_2$ , and

$$[x_1, x_1 + \lfloor \sqrt{x_1} \rfloor], [x_2, x_2 + \lfloor \sqrt{x_2} \rfloor] \in \mathcal{C}_i(p).$$

Without loss of generality, we assume that  $x_1, x_2$  are the largest and smallest numbers satisfying the above conditions.

We now argue that  $[x, y]$  must contain  $[x_2, x_2 + \lfloor \sqrt{x_2} \rfloor]$ . First, we note that

$$\begin{aligned} \sqrt{x} &= (1 + o(1))\sqrt{p/2^{i+1}}, \\ \sqrt{x_2} &= (1 + o(1))\sqrt{p/2^{i+1}}, \text{ and,} \\ \sqrt{x_2} &= (1 + o(1))\sqrt{x}. \end{aligned}$$

By elementary algebra, we have

$$x_2 - x_1 \leq \left(\frac{1}{3} + o(1)\right) \sqrt{\frac{p}{2^{i+1}}},$$

which implies that

$$\begin{aligned} x_2 - x &\leq \left(\frac{1}{3} + o(1)\right) \sqrt{\frac{p}{2^{i+1}}}, \text{ which implies,} \\ x_2 &\leq x + \left(\frac{1}{3} + o(1)\right) \sqrt{x}. \end{aligned}$$

Thus, we have,

$$\begin{aligned} x_2 + \lfloor \sqrt{x_2} \rfloor &\leq x + \left(\frac{4}{3} + o(1)\right) \sqrt{x}, \\ &\leq y, \end{aligned}$$

for  $p/2^{i+1}$  sufficiently large. ■

**A further property of  $\mathcal{C}_i(p)$ .**

Lemma 2.7 is crucial to our analysis. Now, instead of having to show that

$$|\mathcal{I}_i(p)| = O\left(\sqrt{p/2^{i+1}}\right),$$

intervals all have sufficiently many primes in them, we only have to show that

$$|\mathcal{C}_i(p)| = O(1),$$

intervals have sufficiently many primes. We have paid a price for this, since the intervals in  $\mathcal{C}_i(p)$  are smaller than the intervals in  $\mathcal{I}_i(p)$ , but this turns out not to be a problem. To facilitate later arguments, we extend our notion of sparseness to  $\mathcal{C}_i(p)$ .

**Definition 2.13** Let  $p$  be a prime. We say that  $\mathcal{C}_i(p)$  is *sparse* if any of the intervals in  $\mathcal{C}_i(p)$  is sparse.

We need one more property before we use the theorem of Heath-Brown to complete the proof of Theorem 2.9. Our intuition is as follows. If any of the intervals in  $\mathcal{I}_i(p)$  do not contain enough primes, we will be unable to

show that our algorithm runs fast on  $p$ . Heath-Brown show that only a vanishing fraction of the intervals of the form  $[x, x + \lfloor \sqrt{x} \rfloor]$  will not have enough primes. But what if these “bad” intervals appear in most of the  $\mathcal{C}_i(p)$ ’s? Conceivably, the bad intervals could destroy a disproportionate number of primes. Fortunately, the following lemma shows that this is not the case.

**Lemma 2.8** Let  $x$  be sufficiently large. Then an interval of the form,

$$[x, x + \lfloor \sqrt{x} \rfloor],$$

can be in  $\mathcal{C}_i(p)$  for at most  $c \cdot 2^i$  different values of  $p$ , where  $c$  is a constant.

**Proof:** It suffices to show that there are, for each value of  $k \in [-22, 22]$ , only  $O(2^i)$  values of  $p$  which satisfy the equation  $\lfloor f_k(p) \rfloor = x$ , where

$$f_k(p) = \frac{p}{2^{i+1}} + \frac{k}{3} \cdot \sqrt{\frac{p}{2^{i+1}}}.$$

We first eliminate the integer rounding by noting that

$$\lfloor z \rfloor = x \implies x - 1 \leq z \leq x + 1.$$

We therefore have to show that there only  $O(2^i)$  integers  $p$  which satisfy

$$x - 1 \leq f_k(p) \leq x + 1.$$

Therefore, if  $[x, x + \lfloor \sqrt{x} \rfloor]$  is in  $\mathcal{C}_i(p)$  and  $\mathcal{C}_i(p')$ , then  $|f(p') - f(p)| \leq 2$ . We will use this fact to show that  $p$  and  $p'$  must be near to each other in value, which will in turn give us the desired bound. For the rest of the proof, we assume without loss of generality that  $p \leq p'$ .

Let us consider  $f$  in the continuous domain. For all  $p > 0$ , the derivative  $f'(p)$  is at least  $2^{-(i+1)}$ . Since  $f$  is clearly monotone increasing, we have  $f(p') - f(p) \leq 2$ . By elementary calculus, we have

$$f(p') - f(p) \geq \frac{p' - p}{2^{i+1}},$$

from which we can derive,

$$p' - p \leq 2 \cdot 2^{i+1}.$$

This clearly implies that only  $c \cdot 2^i$  solutions exist, for some constant  $c$ . ■

### The final calculation.

We now give the final calculations for bounding the number of primes for which our algorithm will fail. First, we argue that the number of  $k$ -bit primes  $p$  such that  $\mathcal{C}_i(p)$  will be sparse will be small, where  $c$  is a positive constant.

**Lemma 2.9** Let  $2^{k-i}$  be sufficiently large. Then there will be at most

$$\frac{2^k}{2^{1/5(k-i)}}$$

$k$ -bit primes  $p$  such that  $\mathcal{C}_i(p)$  is sparse.

**Remark:** Here,  $1/5$  may be replaced by any number less than  $1/6$ .

**Proof:** First, we note that if  $[x, x + \lfloor \sqrt{x} \rfloor]$  is in  $\mathcal{C}_i(p)$  for  $p \in [2^{k-1}, 2^k]$ , then  $x \in [2^{k-i-2}, 2^{k-i+1}]$ . This follows from the definition of  $\mathcal{C}_i(p)$  and the bounds on  $p$ . We now use Heath-Brown's theorem on the intervals  $[2^{k-i-2}, 2^{k-i-1}]$ ,  $[2^{k-i-1}, 2^{k-i}]$ , and  $[2^{k-i}, 2^{k-i+1}]$ , and sum the results. This bounds the number of sparse intervals in

$$\bigcup_{p \in [2^{k-1}, 2^k]} \mathcal{C}_i(p),$$

to at most,

$$\sum_{j=0}^2 2^{5/6(k-i-j)} \log^\alpha 2^{k-i-j} \leq c_1 \cdot 2^{5/6(k-i)} \log^\alpha 2^{k-i},$$

for some constant  $c_1$ . By Lemma 2.8, each sparse interval of this form is in  $\mathcal{C}_i(p)$  for at most  $c_2 2^i$  different values of  $p$ , where  $c_2$  is some constant. Thus, at most

$$\begin{aligned} (c_2 \cdot 2^i) (c_1 \cdot 2^{5/6(k-i)} \log^\alpha 2^{k-i}) &= c_1 c_2 \frac{2^k \log^\alpha 2^{k-i}}{2^{1/6(k-i)}} \\ &\leq \frac{2^k}{2^{1/5(k-i)}}, \end{aligned}$$

for  $2^{k-i}$  sufficiently large. ■

We can now upper-bound the number of  $k$ -bit primes that PROVE-PRIME will not quickly certify as prime. In order for a prime  $p$  to not be quickly

certified, as per the analysis of Theorem 2.7, it must be the case that  $\mathcal{C}_i(p)$  is sparse for some value of  $i$ . Furthermore, the value of  $i$  must be sufficiently small that  $\text{PROVE-PRIME}(p)$  could, with nonzero probability, proceed for  $i$  steps without  $p_i$  being so small as to be verified deterministically. We denote by  $i_k$  the greatest number of reduction steps the algorithm could possibly go through on a  $k$ -bit prime. Using Lemma 2.9, we bound the number of  $k$ -bit primes that could conceivably not be quickly certified by

$$\begin{aligned} |\{p \in [2^{k-1}, 2^k], p \text{ isn't quickly certified}\}| &\leq \sum_{j=1}^{i_k} \frac{2^k}{2^{1/5(k-i)}}, \\ &\leq \frac{c \cdot 2^k}{2^{1/5(k-i_k)}}, \end{aligned}$$

for some constant  $c$ , by standard properties of geometric series.

We can use Lemma 2.6 to bound below the value of  $2^{k-i_k}$ . Suppose that, on some  $k$ -bit prime  $p$ , the primality proving algorithm proceeded for  $i_k$  reductions before hitting its last prime,  $p_{i_k}$ . Recall that, for  $k$  sufficiently large, the algorithm will stop as soon as

$$p_{i_k} \leq 2^{(\lg p)^{C/\lg \lg \lg p}},$$

We also have,

$$p_{i_k-1} \geq 2^{(\lg p)^{C/\lg \lg \lg p}},$$

or the algorithm would have stopped after the  $(i_k - 1)$ th reduction. Since,  $p_{i_k} \leq p_{i_k-1}/3$ , for  $p_{i_k-1}$  sufficiently large (to give a very weak bound), we have,

$$p_{i_k} \geq \frac{1}{3} \cdot 2^{(\lg p)^{C/\lg \lg \lg p}},$$

for  $k$  sufficiently large. Since  $p_{i_k} = p/2^{i_k} + O(\sqrt{p/2^{p(i)}})$ , by Lemma 2.6, and  $k - 1 \leq \lg p \leq k$ , we have,

$$\lg p_{i_k} \leq k - i_k + 1,$$

for  $k$  sufficiently large. Hence, we have

$$2^{k-i_k} \geq \frac{1}{6} \cdot 2^{(\lg p)^{C/\lg \lg \lg p}}.$$

Since the right hand side of the equation is monotone increasing, we have

$$2^{k-i_k} \geq \frac{1}{6} \cdot 2^{(k-1)^{C'/\lg \lg(k-1)}}.$$

Using this inequality, we have

$$\begin{aligned} \frac{c \cdot 2^k}{2^{1/5(k-i_k)}} &\leq c' \cdot 2^k / 2^{\frac{1}{5}(k-1)^{C'/\lg \lg(k-1)}}, \\ &\leq 2^k / 2^{(k)^{C'/\lg \lg k}}, \end{aligned}$$

for some suitably chosen  $C'$ . ■

## Chapter 3

# Committing Bits Using Oblivious Transfer

### 3.1 Introduction.

In this chapter we discuss techniques for committing bits, and committing bits with zero-knowledge proofs about these committed bits. These techniques all make use of an ideal oblivious transfer channel. We begin by giving some intuition for just what bit commital is, and why it is so useful in protocol design.

#### What is bit commital?

The essence of bit commital can be understood by considering the following story. Alice and Bob wanted to flip a fair coin, but had no physical coin to flip. Alice offered a simple way of flipping a fair coin mentally. “First, you think up a random bit, then I’ll think up a random bit. We’ll then exclusive-or the two bits together.” she suggested.

“But what if one of us doesn’t flip a coin at random?” Bob asked.

“It doesn’t matter. As long as one of the bits is truly random, the exclusive-or of the bits should be truly random.” Alice replied, and after a moment’s reflection, Bob agreed.

A short while later, Alice and Bob happened upon a book on artificial intelligence, lying abandoned by the roadside. A good citizen, Alice said,

“One of us must pick this book up, and find a suitable waste receptacle.” Bob agreed, and suggested they use their coin-flipping protocol to determine who would have to throw the book away.

“If the final bit is a 0, then you will pick the book up, and if it is a 1, then I will.” said Alice. “What is your bit?”

Bob replied, “1.”

“Why, so is mine,” said Alice, slyly, “I guess this isn’t your lucky day.”

Needless to say, this coin flipping protocol had a serious bug. While it is true that a truly random bit,  $x$ , exclusive-ored with any *independently* distributed bit,  $y$ , will yield a truly random bit, Alice’s protocol did not ensure that the two bits were distributed independently. In fact, it is not hard to verify that no mental protocol can allow two infinitely powerful parties to flip a fair coin. Alice and Bob were in a quandary, until they received a letter from an obscure graduate student in cryptography. The information in the letter was too theoretical to be of any earthly use to anyone, but the envelope the letter came in was extremely handy.

The next time Alice and Bob wished to flip a coin, they played a modified version of the original protocol. First, Bob decided on a bit, but instead of announcing it immediately, he wrote it down on a piece of paper, and placed the paper in the envelope. Next, Alice announced her bit. Finally, Alice and Bob took Bob’s bit out of the envelope, and computed the random bit. This bit was indeed truly random whenever at least one of them played honestly. Alice and Bob had a working protocol, the cryptographer’s dream of social relevance was fulfilled, and they all lived happily ever after.

In its simplest form, commital is nothing more than simulating the envelope in the above story. Indeed, in much of our intuitive reasoning for why protocols work, we will appeal directly to the envelope analogy.

## **Commital with zero-knowledge proofs.**

In this thesis, we will actually use a much more powerful notion of commitment, which we dub, “Commital with zero-knowledge proof.” To illustrate what we mean by this, we consider the following Alice-Bob story.

One day, Bob was desperately trying to factor a 500 bit number,  $n$  which he knew was a product of five 100 bit numbers. Alice walked by, and said, “I happen to know one of the factors of  $n$ , which I’ll gladly give to you for \$100.” To show that she was serious, she took out one hundred envelopes,

and proceeded to commit one hundred bits for Bob. Unfortunately, Bob had only \$25. Unwilling to cut her rates, Alice offered to sell Bob 25 bits of the number she had committed.

“But how will I know that the number you committed is actually a factor of  $n$ ?” Bob asked. “If you would show me the number first, and let me verify that it is a factor, I’ll agree to your terms.”

Naturally, Alice did not go for this deal. This was a serious quandary, which even envelopes did not appear to solve. Alice could not convince Bob her number was any good without revealing it, and Bob would not buy 25 bits of a number that might well be random garbage. Is there a way out of this difficulty?

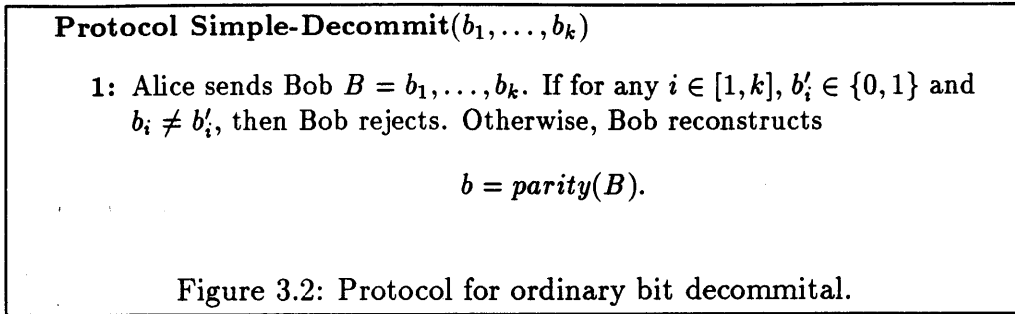
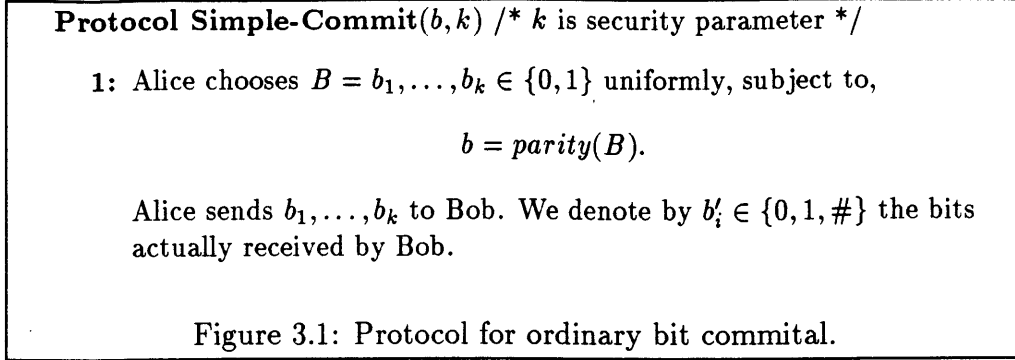
In this chapter we will show how, using oblivious transfer, one can commit a set of bits, and somehow prove to the other party that these committed bits has some special property.

### 3.1.1 Outline of the Chapter.

In Section 3.2, we give some very simple commital protocols based on oblivious transfer, due to Crépeau. In Section 3.3, we show how to efficiently commit elements of a finite group, along with a zero-knowledge proof that these elements have a certain product. In Section 3.4, we extend this result to allow us to commit a set of bits, along with a zero-knowledge proof of an arbitrary  $NP$  assertion about these bits. In Section 3.5, we explicitly write down all the simulators for our protocols, along with formal arguments of their correctness. Finally, in Section 3.6, we observe that our schemes can be strengthened by running them in parallel.

## 3.2 A simple commital protocol using oblivious transfer.

In this section, we outline simple protocols, due to Crépeau, for committing and decommitting a bit using an oblivious transfer channel. We suppose Alice wishes to commit a bit,  $b$ , to Bob, and then decommit it at a later date. For convenience, we define the following two functions.



**Definition 3.1** Given a  $k$ -bit bit sequence  $B = b_1, \dots, b_k$ , we define  $\text{parity}(B) = b_1 \oplus \dots \oplus b_k$ .

**Definition 3.2** We define  $\text{majority}(b_1, \dots, b_k)$  as equal to 1 iff

$$|\{i : b_i = 1\}| \geq |\{i : b_i = 0\}|.$$

We now wish to argue that these protocols, given Figures 3.1 and 3.2, give some of the same security guarantees as do physical envelopes, albeit in a weaker form. Specifically, we wish to show that,

1. (Security) Suppose Bob has no *a priori* information about  $b$ . Then after Alice has committed  $b$ , but before Alice has decommitted  $b$ , There is only a  $1/2^k$  chance that Bob will be able to guess  $b$  with probability greater than  $\frac{1}{2}$ .

2. (Confidence) The probability that Alice can trick Bob into reconstructing a bit different than the bit she originally committed is at most  $\frac{1}{2}$ .

**Lemma 3.1** [Security] At the end of  $\text{SIMPLE-COMMIT}(b, k)$ , the probability that Bob can predict  $b$  any better than before is  $1/2^k$ .

**Proof:** With probability  $1/2^k$ , Bob receives  $b_1, \dots, b_k$  entirely. In other words,  $b'_i = b_i$  for  $1 \leq i \leq k$ . In this case, Bob can easily reconstruct  $b = \text{parity}(B) = b'_1 \oplus \dots \oplus b'_k$ . Otherwise,  $b'_i = \#$  for some  $i$ . By symmetry, we consider only consider the case where  $i = k$ . We now argue that knowing  $b_1, \dots, b_{k-1}$  gives no information about  $b$ . This follows from the fact that for all possible values of  $b_1, \dots, b_{k-1}$ , and for either value of  $b$ , there is exactly one value of  $b_k$  that satisfies  $b = b_1 \oplus \dots \oplus b_k$ . ■

This argument is merely a recapitulation of the fact that knowing a proper subset of a set of coin flips tells one nothing about the parity of the entire set of coin flips.

**Lemma 3.2** [Reliability] Suppose that Alice sends Bob the sequence  $B = b_1, \dots, b_k$ . Let  $b = \text{parity}(B)$ . For any strategy Alice may employ, the probability that, at the end of  $\text{SIMPLE-DECOMMIT}$ , Bob reconstructs a bit  $b' \neq b$  is at most  $\frac{1}{2}$ .

**Proof:** Let  $\hat{B} = \hat{b}_1, \dots, \hat{b}_k$  denote the string Alice sends Bob in protocol  $\text{SIMPLE-DECOMMIT}$  (Where Alice is possibly cheating). If  $\hat{B} = B$ , then Bob will accept and reconstruct  $b$ . Otherwise, for some  $i$ ,  $\hat{b}_i \neq b_i$ . However, the distribution of  $\hat{B}$  is independent of the set  $\text{received} = \{j : b'_j \neq \#\}$ , by the obliviousness property of oblivious transfer<sup>1</sup>. Thus, with probability  $\frac{1}{2}$ ,  $b'_i = b_i \neq \hat{b}_i$ , and Bob will reject. ■

### 3.2.1 Improving the Committal/Decommital Protocols.

We must reconcile the nonideal behavior of the commital/decommital protocols outlined above with the ideal behavior we would like to simulate. In

---

<sup>1</sup>We assume in this proof that Bob does not give Alice any information about which bits he has received. This will indeed be the case in all the larger protocols our commital protocols are based on.

**Protocol Secure-Simple-Commit( $b, k$ )** /\*  $k$  is security parameter \*/

- 1: Alice chooses a sequence of bit sequences  $\mathcal{B} = B^1, \dots, B^k \in \{0, 1\}^k$ . We denote the bits of these sequences by  $B^i = b_1^i, \dots, b_k^i$ , where  $b_j^i \in \{0, 1\}$ . The bit sequences  $B^i$  are chosen uniformly, subject to  $b = \text{parity}(B^i)$ , for  $1 \leq i \leq k$ . Alice transfers  $B^1, \dots, B^k$  to Bob over an oblivious transfer channel. We define  $\mathcal{B}' = B^{1'}, \dots, B^{k'}$ , where  $B^{i'} = b_1^{i'}, \dots, b_k^{i'}$ , and  $b_j^{i'} \in \{0, 1, \#\}$ , as the bits Bob actually receives from Alice.

Figure 3.3: A strengthened bit commital protocol.

**Protocol Secure-Simple-Decommit( $\mathcal{B}, k$ )** /\*  $\mathcal{B}, \mathcal{B}'$  are the same as in the Commit Protocol \*/

- 1: Alice sends  $\mathcal{B}$  to Bob over a clear channel. If for some  $i, j$ ,  $b_j^{i'} \neq \#$ , and  $b_j^{i'} \neq b_j^i$ , then Bob rejects. Otherwise, Bob reconstructs  $b^{i'} = b_1^{i'} \oplus \dots \oplus b_k^{i'}$ , for  $1 \leq i \leq k$ . If all of the  $b^{i'}$ 's are the same, then Bob reconstructs  $b' = b^{1'}$ .

Figure 3.4: Strengthened decommital protocol.

the physical model for envelopes, the probability that the recipient of a committed bit learns anything about it from the commital process is equal to 0, compared to  $1/2^k$  in our implementation. With envelopes, the probability that Alice can trick Bob into recovering an incorrect bit is likewise equal to 0, compared to  $1/2$  in our implementation. The former discrepancy is not serious, since we consider exponentially small probabilities as being negligible. However, the latter discrepancy is more serious. For example, if the coin flipping protocol described above uses our commital/decommital protocol, then one party can control the output of the “fair” coin fully half the time.

Fortunately, we can fix our protocols by running them several times in parallel, as follows.

Lemmas 3.3 and 3.4 are analogous to Lemmas 3.1 and 3.2.

**Lemma 3.3** At the end of  $\text{SECURE-SIMPLE-COMMIT}(B, K)$ , the probability that Bob can predict  $b$  any better than before is  $k(1/2^k)$ .

**Proof:** The proof follows along the same lines as the proof of Lemma 2.1. The only circumstance in which Bob receives any information about  $b$  is when he receives every bit of one of the bit sequences,  $B^i$ . For any bit sequence  $B^i = b_1^i, \dots, b_k^i$ , the probability that  $b_j^{i'} = b_j^i$ , for all  $j \in [1, k]$ , is  $1/2^k$ . By elementary probability, the probability that one of the  $k$  possible bit sequences is completely received is bounded above by  $k(1/2^k)$ . ■

**Lemma 3.4** Suppose that Alice sends Bob the sequences  $B^1, \dots, B^k$ , where  $B^i = b_1^i, \dots, b_k^i$ , over an oblivious transfer channel. Define  $b^i = \text{parity}(B^i)$ , and  $b = \text{majority}(b^1, \dots, b^k)$ . For any strategy Alice may employ, the probability that, at the end of  $\text{SECURE-SIMPLE-DECOMMIT}$ , Bob reconstructs a bit  $b' \neq b$  is at most  $1/2^{k/2}$ .

**Proof:** Let  $\hat{B}^1, \dots, \hat{B}^k$  denote the strings Alice sends Bob in protocol  $\text{SECURE-SIMPLE-DECOMMIT}$  (Where Alice is possibly cheating). We write  $\hat{B}^i = \hat{b}_1^i, \dots, \hat{b}_k^i$ . By our definition of  $b$ , we have that for at least  $k/2$  of the  $i$ 's in  $[1, k]$ ,  $b^i = b$ . Now, if for any  $i$ ,  $\text{parity}(\hat{B}^i) = b$ , then Bob will either reject or recover  $b' = b$ . Therefore, the only way Bob can reconstruct  $b' \neq b$  is if for at least  $k/2$  of the  $i$ 's in  $[1, k]$ ,  $\hat{B}^i \neq B^i$ . It then follows that there exist  $\lceil k/2 \rceil$  pairs,  $(i_1, j_1), \dots, (i_{\lceil k/2 \rceil}, j_{\lceil k/2 \rceil})$ , such that

$$\hat{b}_{j_m}^{i_m} \neq b_{j_m}^{i_m},$$

for  $1 \leq m \leq \lceil k/2 \rceil$ . Thus, by the same argument as in the proof of Lemma 3.2, the probability that Bob rejects is at least  $1 - 2^{-\lceil k/2 \rceil}$ . ■

In general, our strategy for developing commital protocols is to ensure that

1. With probability exponentially close to 1, the recipient of a committed bit will not receive any information about the committed bit.
2. For some  $c$ , the probability of successful cheating (tricking someone into recovering a different bit than what was originally committed to) is bounded above by  $1 - 1/k^c$ .

We then strengthen the second condition by running the commital protocol in parallel.

### 3.3 Proving Assertions about Products of Group Elements.

So far, we have shown how to commit and decommit bits using oblivious transfer. This by itself is a very useful tool in protocol design, but in fact we can do much better. In the rest of the chapter, we extend these techniques, and implement a form of bit commitment which allows one to give zero-knowledge proofs of assertions about the committed bits. These protocols will form the foundation for many of our more complicated protocols.

In order to motivate and explain our zero-knowledge commitment scheme, we first consider a seemingly much different problem. Suppose that Alice has a sequence,  $A = a_1, \dots, a_k$ , of group elements for some group  $G$ . She is willing to let Bob learn a small number of these elements, say  $\log k$  or  $k^{1/4}$  of them, and is also willing to let him learn their product,  $a = a_1 a_2 \dots a_k$ . However, she doesn't want him to learn anything that cannot be inferred from knowing this product and the values of a small subset of the elements.

We allow Alice to send group elements to Bob through a special group element oblivious transfer channel. This channel takes as input a group element  $g$ , and sends it to Bob with some probability  $p$  (otherwise, sending the “not sent” symbol, ‘#’). The value of  $p$  is specified by the protocol being executed. We adopt the convention that the first  $k$  elements Alice attempts to send Bob through this channel define her values,  $a_1, \dots, a_k$ .

Alice would like to give Bob some confidence that the product  $a_1 \dots a_k$  is indeed equal to  $a$ . Specifically, we want there to exist some constant,  $c$ , such that if Alice were lying about the value of the group product, Bob would catch her with probability at least  $1/k^c$ .

Finally, we do not allow Bob to communicate with Alice.

#### 3.3.1 An incorrect solution.

To motivate our solution to this problem, we first give an incorrect solution, which illustrates one of our main ideas. For ease of exposition, we assume that  $k$  is a perfect power of 2, and write  $k = 2^m$ . We first make the following definition.

**Definition 3.3** We define a *condenser tree*,  $C$ , for  $A = a_1 \dots, a_k$ , as follows. We denote the elements of  $C$  by  $c_{i,j}$ , where  $1 \leq i \leq m$ , and  $1 \leq j \leq 2^{m-i}$ .

1. For  $1 \leq j \leq k/2$ ,  $c_{1,j} = a_{2j-1}a_{2j}$ .
2. For  $2 \leq i \leq m$  and  $1 \leq j \leq 2^{m-i}$ ,  $c_{i,j} = c_{i-1,2j-1}c_{i-1,2j}$ .

By a trivial induction, we have the following invariance property.

**Lemma 3.5** Let  $C = \{c_{i,j}\}$  be the condenser tree for  $A = a_1 \dots, a_k$ . Then,

$$(\forall i, 1 \leq i \leq m) \prod_{j=1}^{2^{m-i}} c_{i,j} = \prod_{j=1}^k a_j. \quad \blacksquare$$

In particular,  $c_{m,1} = \prod_{i=1}^k a_i$ .

Using the above lemma, we construct the protocol given in Figure 3.5. It should be noted that the transfer probability,  $p = 1/k^3$ , is arbitrarily chosen to be small, but nonnegligible. Setting  $p = 1/k^c$  for any  $c$  sufficiently large would also suffice.

The following lemma demonstrates that some confidence about the value of  $\prod_{i=1}^k a_i$  is transferred to Bob.

**Lemma 3.6** In protocol NAIVE-PROVE-PRODUCT( $A, k$ ), suppose that

$$a \neq \prod_{i=1}^k a_i,$$

where  $a$  and the  $a_i$ 's are defined as in Step 1 of the protocol. Then, for any value of  $C$  which Alice (possibly cheating) sends in Step 2 of the protocol, the probability that Bob accepts is at most  $(1 - 1/k^9)$ .

**Proof:** Suppose that  $C = \{c_{i,j}\}$  is the condenser tree for  $A = a_1, \dots, a_k$ . Then  $c_{m,1} = a_1 a_2 \dots a_k \neq a$ . With probability  $1/k^3$ ,  $c'_{m,1} = c_{m,1} \neq a$ , and Bob will reject. If  $C$  is not a valid condenser tree for  $A$ , then one of the following two conditions must hold.

1.  $(\exists i, j) c_{i,j} \neq c_{i-1,2j-1}c_{i-1,2j}$ .
2.  $(\exists j) c_{1,j} \neq a_{2j-1}a_{2j}$ .

In case 1, with probability  $1/k^9$ ,

$$c'_{i,j}, c'_{i-1,2j-1}, c'_{i-1,2j} \neq \#, \text{ and} \\ c'_{i,j} \neq c'_{i-1,2j-1}c'_{i-1,2j}.$$

**Protocol Naive-Prove-Product(A,k)**

- 1: Let  $A = a_1, \dots, a_k$ . We set the transfer probability,  $p$ , of the group element oblivious transfer channel, to  $1/k^3$ . Alice transfers  $A$  to Bob through the channel. We denote by  $A' = a'_1, \dots, a'_k$  the elements Bob actually receives. She also sends  $a = \prod_{i=1}^k a_i$  to Bob through a clear channel.
- 2: Let  $C = \{c_{i,j}\}$  be the condenser tree for  $A$ . Alice transfers  $C$  to Bob through the channel. We denote by  $C' = \{c'_{i,j}\}$  the elements Bob actually receives.

**Checking Step:** Bob makes the following checks

- If  $c'_{i,j}, c'_{i-1,2j-1}, c'_{i-1,2j} \neq \#$ , and  $c'_{i,j} \neq c'_{i-1,2j-1}c'_{i-1,2j}$ , then Bob rejects.
- If  $c'_{1,j}, a'_{2j-1}, a'_{2j} \neq \#$ , and  $c'_{1,j} \neq a'_{2j-1}a'_{2j}$ , then Bob rejects.
- If  $c'_{m,1} \neq \#$ , and  $c'_{m,1} \neq a$ , then Bob rejects.

Figure 3.5: A naive scheme for proving assertions about permutation products.

In case 2, with probability  $1/k^9$ ,

$$c'_{1,j}, a'_{2j-1}, a'_{2j} \neq \#, \text{ and} \\ c'_{1,j} \neq a'_{2j-1} a'_{2j}.$$

Thus, no matter what Alice does, Bob rejects with probability  $1/k^9$ . ■

## Local Verifiability

The crux of the above proof is that if a tree is not a valid condenser tree, one can prove that it is invalid by considering the values of only a constant number of the tree's nodes. Conversely, one can verify that a condenser tree is correct by executing a series of local checks, each of which only looks at a constant number of nodes (in this case, three). We call this property *local verifiability*. If a property can be verified through a series of local checks, where each test looks at most  $k$  locations, then we say the property is  $k$ -locally verifiable.

In general, if a structural property,  $P$ , is  $k$ -locally verifiable, then one can transfer some confidence that  $P(X)$  holds for some structure  $X$ , without revealing very much information about  $X$ . Suppose that one could obviously transfer each component of  $X$  to a verifier, and that verifier's chance of receiving each component is  $1/n^c$ . Then, if  $P(X)$  did not hold, there would be a set of  $k$  components which, if received by the verifier, would demonstrate this fact. Thus, the verifier can reject an invalid (with respect to  $P$ ) structure  $X$  with probability  $(1/n^c)^k = 1/n^{ck}$ . Conversely, if the verifier doesn't reject  $X$ , then it has obtained a nonnegligible amount of confidence that  $P(X)$  holds. Note that this transfer of confidence may occur even when with high probability the verifier only receives a very small glimpse at the components of  $X$ . This "paradox" makes our advanced commitment protocols possible.

## Why doesn't this scheme work?

The above protocol transfers some confidence to Bob about the product  $a_1 a_2 \dots a_k$ . Furthermore, the probability that Bob receives *any* of the group elements transferred in the entire protocol is  $O(1/k^2)$ . Thus, most of the time, Bob gets absolutely no information about  $A$ , other than its product. The protocol we described above would seem to fit the bill. However, there is

a catch. With probability  $1/k^3$ , Bob learns the value of  $c_{m-1,1} = a_1 a_2 \dots a_{k/2}$ . This information clearly cannot be derived from knowing just a few of the  $a_i$ 's and the product of all the  $a_i$ 's. In some sense, our information is too tightly coupled. Intuitively, information about the original set of group elements is being compressed, through successive multiplications, into a smaller and smaller set of group elements. The final group element contains information about all of the original elements, so it is no surprise that some of the elements before it also contain highly nonlocal information. To solve this problem, we introduce a randomizing step, which we describe below.

### 3.3.2 Randomizing Tableaus.

Given a sequence,  $A = a_1, \dots, a_k$ , of group elements, we wish to, through a sequence of local changes, create a new sequence,  $A^R = a_1^R, \dots, a_k^R$ , such that

$$a_1 a_2 \dots a_k = a_1^R a_2^R \dots a_k^R.$$

To do this, we define a structure we call a *randomizing tableau*. We first define a function,  $change(n, k)$ , which will specify the locations of the local changes we will make, and the notion of a *randomization sequence*.

**Definition 3.4** We define the function  $change(k, n)$  by

$$change(k, n) = remainder(n, k - 1) + 1,$$

where  $remainder(m, n)$  is the smallest nonnegative number congruent to  $m$  mod  $n$ .

**Definition 3.5** Let  $G$  be a group. We formally define a *randomizing sequence* of length  $n$  to be a sequence  $r_1, \dots, r_n$ , where  $r_i \in G$ .

**Definition 3.6** Let  $G$  be a group, and let  $A = a_1, \dots, a_k$ , where  $a_i \in G$ . Let  $R = r_1, \dots, r_n$  be a randomization sequence for  $G$ . An *n-stage randomizing tableau*,  $T(A, R)$ , is an  $n + 1$  by  $k$  matrix, defined as follows. We write  $T(A, R) = [t_{i,j}]$ , where  $0 \leq i \leq n$  and  $1 \leq j \leq k$ .

1.  $t_{0,j} = a_j$ .

2. For  $i > 0$ ,  $t_{i,j}$  is defined by,

$$t_{i,j} = \begin{cases} t_{i-1,j} r_i^{-1} & j = \text{change}(i, k), \\ r_i t_{i-1,j} & j = \text{change}(i, k) + 1, \\ t_{i-1,j} & j \neq \text{change}(i, k), \text{change}(i, k) + 1. \end{cases}$$

**Notation:** As we hinted before, one can view the rows of a randomizing tableau as a sequence of strings obtained by starting with string  $A$ , and making a sequence of local changes. To support this viewpoint, we denote by  $A_i^R(T)$  the  $i$ th row of tableau  $T = T(A, R)$ , i.e.,

$$A_i^R(T) = t_{i,1}, \dots, t_{i,k}.$$

We also define the final row,  $A_n^R(T)$  to be  $A^R(T)$ . In cases where  $T$  is clear, we drop it from our notation, simplifying  $A_i^R(T)$  to  $A_i^R$  and  $A^R(T)$  to  $A^R$ .

### An example.

Let our group  $G$  be  $Z_3$ . Let  $A = 1, 2, 0, 1$ , and  $R = 2, 1, 0, 2, 2, 1$ . Then  $T(A, R)$  is equal to

$$\begin{array}{rcl} 0 & 0 & 2 \ 2 = A_6^R = A^R \\ 0 & 0 & 0 \ 1 = A_5^R \\ 0 & 2 & 1 \ 1 = A_4^R \\ 2 & 0 & 1 \ 1 = A_3^R \\ 2 & 0 & 1 \ 1 = A_2^R \\ 2 & 1 & 0 \ 1 = A_1^R \\ 1 & 2 & 0 \ 1 = A_0^R = A \end{array}$$

We use slightly nonstandard notation by writing the rows in decreasing order instead of increasing. Intuitively, we view row 0 as the base of the tableau, and row  $n$  as the top of the tableau.

### 3.3.3 Four Properties of Randomizers.

We now establish four key properties of randomizers. Two of these properties demonstrate how one may use randomizing tableaux to transfer confidence about certain facts, and the other two bound how much information is transferred under certain circumstances.

## Transferring Confidence

The two properties possessed by randomizing tableaux which make them amenable to confidence transfer are local verifiability and *row-product invariance*. As was the case with condenser trees, randomizing tableaux are locally verifiable, as shown in Lemma 3.7. Row-product invariance refers to the fact that the products of the rows of a randomizing tableau are all the same. We prove this property in Lemma 3.9.

**Lemma 3.7** [local verifiability] Given a sequence  $A$  and a matrix  $T$ , one can prove that

$$(\forall R)T \neq T(A, R)$$

by considering only 4 elements of  $A \cup T$ .

**Proof:** We first prove an alternate criterion for when an  $R$  exists such that  $T = T(A, R)$ .

**Lemma 3.8** Given a sequence  $A = a_1, \dots, a_k$  and  $n + 1$  by  $k$  matrix  $T = [t_{i,j}]$ , there exists a randomizing sequence  $R = r_1, \dots, r_n$  such that

$$T = T(A, R)$$

iff  $(\forall i, 1 \leq i \leq n)(\forall j, 1 \leq j \leq k)$ ,

1.  $t_{0,j} = a_j$ .
2.  $t_{i,j} = t_{i-1,j}$ , for  $j \neq \text{change}(i, k), \text{change}(i, k) + 1$ .
3.  $t_{i,j}t_{i,j+1} = t_{i-1,j}t_{i-1,j+1}$ , for  $j = \text{change}(i, k)$ .

**Proof:** ( $\Leftarrow$ ) If  $T = T(A, R)$  for some  $R$ , then conditions 1 and 2 follow immediately from the definition of randomizing tableaux. For  $1 \leq i \leq n$ , and  $j = \text{change}(i, k)$ , we have

$$t_{i,j} = t_{i-1,j}r_i^{-1}, \text{ and}$$

$$t_{i,j+1} = r_i t_{i-1,j+1}.$$

But this implies that

$$\begin{aligned} t_{i,j}t_{i,j+1} &= (t_{i-1,j}r_i^{-1})(r_i t_{i-1,j+1}) \\ &= t_{i-1,j}(r_i^{-1}r_i)t_{i-1,j+1} \\ &= t_{i-1,j}t_{i-1,j+1} \end{aligned}$$

( $\implies$ ) We prove this direction by explicitly constructing such an  $R$ . We define  $R = r_1, \dots, r_n$  by

$$r_i = t_{i,j}^{-1} t_{i-1,j},$$

where  $j = \text{change}(i, k)$ . We now verify that,

1.  $t_{i,j} = t_{i-1,j} r_i^{-1}$ , and,
2.  $t_{i,j+1} = r_i t_{i-1,j}$ ,

for  $1 \leq i \leq n$ , and  $j = \text{change}(i, k)$ . We have

$$\begin{aligned} t_{i-1,j} r_i^{-1} &= t_{i-1,j} (t_{i-1,j}^{-1} t_{i,j}) \\ &= (t_{i-1,j} t_{i-1,j}^{-1}) t_{i,j} \\ &= t_{i,j}. \end{aligned}$$

Similarly, since  $t_{i,j} t_{i,j+1} = t_{i-1,j} t_{i-1,j+1}$ , for  $j = \text{change}(i, k)$ , we have

$$\begin{aligned} t_{i,j+1} &= t_{i,j}^{-1} (t_{i-1,j} t_{i-1,j+1}) \\ &= (t_{i,j}^{-1} t_{i-1,j}) t_{i-1,j+1} \\ &= r_i t_{i-1,j}. \quad \blacksquare \end{aligned}$$

To prove the main lemma, we simply note that our new criterion for when  $(\exists R)T = T(A, R)$  may be written as a set of  $O(nk)$  conditions, each of which refer to at most 4 locations in  $A$  and  $T$ .  $\blacksquare$

As was the case with condenser trees, the local verifiability of randomizing tableaux makes them very useful building blocks in our protocols. By transferring only a small expected number of entries in a purportedly valid tableau, one can transfer some confidence that it is indeed valid.

**Lemma 3.9** [row-product invariance] Let  $A = a_1, \dots, a_k$  be a sequence of group elements, and  $R = r_1, \dots, r_n$  be a randomization sequence. Writing  $T(A, R) = [t_{i,j}]$ , we have for  $0 \leq i \leq n$ ,

$$\prod_{j=1}^k t_{i,j} = \prod_{j=1}^k a_j.$$

**Proof:** It suffices to prove the following two assertions.

1.  $\prod_{j=1}^k t_{0,j} = \prod_{j=1}^k a_j$ .
2.  $\prod_{j=1}^k t_{i,j} = \prod_{j=1}^k t_{i-1,j}$ , for  $i > 0$ .

The first assertion follows immediately from the identity,  $t_{0,j} = a_j$ . The second assertion follows almost as easily from our definition of  $T$ . Let  $m = \text{change}(i, k)$ . We can write  $A_i^R$ , the  $i$ th row of  $T$ , as

$$A_i^R = t_{i-1,1}, \dots, t_{i-1,m-1}, t_{i-1,m} r_i^{-1}, r_i t_{i-1,m+1}, t_{i-1,m+2}, \dots, t_{i-1,k}.$$

Thus, we have

$$\begin{aligned} \prod_{j=1}^k t_{i,j} &= \left( \prod_{j=1}^{m-1} t_{i-1,j} \right) (t_{i-1,m} r_i^{-1}) (r_i t_{i-1,m+1}) \left( \prod_{j=m+2}^k t_{i-1,j} \right) \\ &= \left( \prod_{j=1}^{m-1} t_{i-1,j} \right) (t_{i-1,m} t_{i-1,m+1}) \left( \prod_{j=m+2}^k t_{i-1,j} \right) \\ &= \prod_{j=1}^k t_{i-1,j}. \end{aligned}$$

The lemma follows. ■

Thus, if one possesses confidence that an  $n + 1$  by  $k$  matrix  $T = [t_{i,j}]$  is a valid randomizing tableau for  $A$ , then one possesses confidence that, in particular,  $t_{n,1} t_{n,2} \dots t_{n,k} = a_1 a_2 \dots a_k$ .

## Bounding knowledge transfer.

We now show that randomizing tableaus have nice properties in regards to how much information one obtains by seeing only a small number of tableau elements. We first show that information about a set of columns in a tableau provides information about only those elements of the base row which are contained in these columns. We then show that if one picks a tableau according to a certain distribution, rows which are far apart in the tableau are only weakly correlated.

Our first lemma, which follows directly from the definition of randomizing tableaus, demonstrates the property we call columnwise independence.

**Lemma 3.10** [columnwise independence] Let  $A = a_1, \dots, a_k$ ,  $R = r_1, \dots, r_n$ ,  $T(A, R) = [t_{i,j}]$ , and  $I \subset [1, k]$ . Then the elements  $\{t_{i,j} | i \in I\}$  are functions of  $R$  and the set of elements  $\{a_i | i \in I\}$ . ■

Thus, having knowledge about the first, fourth, and fifth columns of  $T$  gives one information about  $a_1, a_4, a_5$ , but gives *no* information about  $a_i$  for  $i \neq 1, 4, 5$ . As a consequence of columnwise independence, knowing  $m$  values of  $T$  gives information about at most  $m$  values of  $A$ . This property is in sharp contrast with condenser trees, where a single value of  $C$  could give information about  $O(k)$  elements of  $A$ .

Our second lemma demonstrates the property we call *gap-randomization*. Essentially, it says that if two rows are separated by a sufficiently large gap in a randomly chosen tableau, then they will be essentially independent. They will both have the same row-product, but will otherwise have no correlation.

Before proving this lemma, we first prove a similar lemma which possesses the heart of the proof of the main lemma.

**Lemma 3.11** Fix  $A = a_1, \dots, a_k$ , where  $a_i \in G$ , and  $G$  is finite, and let  $a = \prod_{i=1}^k a_i$ . Let  $R = r_1, \dots, r_{k-1}$  be a uniformly chosen sequence of elements of  $G$ . Let  $\text{scramble}_A(R) = A' = a'_1, \dots, a'_k$  be defined by  $a'_1 = a_1 r_1^{-1}$ ,  $a'_k = r_{k-1} a_k$ , and  $a'_i = r_{i-1} a_i r_i^{-1}$ , for  $1 < i < k$ . Then  $A'$  is uniformly distributed over all  $k$  element sequences whose product is  $a$ .

**Proof:** This follows from the following two assertions.

1. For any choice of  $R$ ,  $a = \prod_{i=1}^k a'_i$ .
2. For any choice of  $A'$ , and with  $A$  fixed, there is exactly one value of  $R$  such that  $A' = \text{scramble}_A(R)$ .

To prove assertion 1, we just write out  $\prod_{i=1}^k a'_i$  as

$$\begin{aligned}
 \prod_{i=1}^k a'_i &= (a_1 r_1^{-1})(r_1 a_2 r_2^{-1}) \cdots (r_{k-1}^{-1}) \\
 &= a_1 (r_1^{-1} r_1) a_2 (r_2^{-1} r_2) \cdots (r_{k-1}^{-1} r_{k-1}) a_k \\
 &= a_1 a_2 \cdots a_k \\
 &= a
 \end{aligned}$$

To prove Assertion 2, we use a counting argument. Denote by  $\mathcal{A}_a^{\parallel}$  the set of  $k$  element sequences of group elements whose product is  $a$ . From our definition, it is not hard to see that  $\text{scramble}_A$  is an injective mapping. Specifically, if  $R$  and  $R'$  first disagree in their  $i$ th element, then  $\text{scramble}_A(R)$  and  $\text{scramble}_A(R')$  will disagree in their  $i$ th element. By assertion 1, we have that the range of  $\text{scramble}_A$  is in  $\mathcal{A}_a^k$ . Clearly,  $|\mathcal{A}_a^k| = |G|^{k-1}$ , since we can choose the first  $k-1$  elements of the sequence arbitrarily, and solve for the unique possible last element. Now, the range of  $\text{scramble}_A$  is also of size  $|G|^{k-1}$ , so the mapping is a bijection. This implies Assertion 2. ■

We can now prove the main lemma.

**Lemma 3.12** [gap-randomness] Let  $G$ , the base group, be finite. Let  $A = a_1, \dots, a_k$ , and let  $R = r_1, \dots, r_n$  be chosen uniformly. Let  $a = \prod_{i=1}^k a_i$ . Then for all  $i$ , the induced distribution on row  $A_{i+k-1}^R$  will be independent from  $A_0^R, \dots, A_i^R$ . That is, given the values of  $A_0^R, \dots, A_i^R$  (the 0th through  $i$ th rows of  $T(A, R)$ ), the conditional distribution of  $A_{i+k-1}^R$  will be uniform over those  $k$  element sequences whose product is equal to  $a$ .

**Proof:** By our notation,  $A_j^R = t_{j,1}, \dots, t_{j,k}$ . Let  $x = \text{change}(k, i)$ , and  $m = i + k - 1$ . By a straightforward calculation, based on our definition of a randomizing tableau, we can solve for the values of  $t_{i+k-1,j}$  ( $1 \leq j \leq k$ ) in terms of the values of  $R$  and the values of  $t_{i+k-1,j}$ , as follows.

$$\begin{aligned}
t_{m,1} &= t_{i,1} r_{m-x+1}^{-1} \\
t_{m,2} &= r_{m-x+1} t_{i,2} r_{m-x+2}^{-1} \\
&\vdots \\
t_{m,x} &= r_{m-1} t_{i,x} r_m^{-1} \\
t_{m,x+1} &= r_m t_{i,x+1} r_{i+1}^{-1} \\
t_{m,x+2} &= r_{i+1} t_{i,x+2} r_{i+2}^{-1} \\
&\vdots \\
t_{m,m-1} &= r_{m-x-1} t_{i,m-1} r_{m-x}^{-1} \\
t_{m,m} &= r_{m-x} t_{i,m}
\end{aligned}$$

The exact indices of the various group elements aren't important. The key observation to be made is that  $A_{i+k-1}^R = \text{scramble}_{A_i^R}(R')$ , where  $R'$  is a

subsequence of  $R$  that has been appropriately shifted with wraparound. The sequence  $R'$  depends only on the values of  $r_{i+1}, \dots, r_{i+k-1}$ . These values are independent of  $A_i^R$ , and are uniformly distributed. Thus, we can apply Lemma 3.11 to yield the desired uniformity condition on the distribution of  $A_{i+k-1}^R$ . ■

Thus, one way to look at randomizing tableaux is to view them as random walks among sequences with a given product. This random walk has the property that no matter where one is located at some stage of the walk, after  $k - 1$  steps, one arrives at a completely random sequence. If, at some point in the walk, someone puts on a blindfold and walks  $k - 1$  steps, one will have no idea where one is. This turns out to be crucial for our purposes.

### Lazy generation of partial randomizing tableaux.

Suppose someone has some base row  $A = a_1, \dots, a_{100}$ , and asks you to generate a 1,000,000-stage randomizing tableau. However, he also tells you that he only wishes to receive rows 1-1000, and rows 999,001-1,000,000 of the tableau. Is there a method of generating these 2,000 rows according to the correct distribution, without generating the other 998,000? In fact, this is easily accomplished using the gap-randomization lemma. First, one generates rows 1-2,000 in the usual manner. Second, one randomly generates row 999,001 uniformly from those whose row-product is equal to  $A$ 's. Finally, one generates the remaining rows 999 in the usual manner. Since the gap between row 1000 and row 999,001 is at least  $100 - 2 = 98$ , and no information is given about any of the intervening rows, the gap-randomness property insures that the lazy method gives the correct distribution on the rows it outputs.

#### 3.3.4 Noninteractively proving product assertions.

We can now give a correct protocol for proving assertions about the product of a sequence of elements of an arbitrary finite group. This protocol, given in Figure 3.6, will with high probability reveal a small number of elements from the sequence. As usual, we work in some finite group,  $G$ .

We wish to prove two properties about this protocol. First, we want to establish that some confidence is transferred about the product of the group

**Protocol Prove-Product(A,k)**

- 1: Let  $A = a_1, \dots, a_k$ . We set the transfer probability,  $p$ , of the group element oblivious transfer channel, to  $1/k^{10}$ . Alice transfers  $A$  to Bob through the channel. We denote by  $A' = a'_1, \dots, a'_k$  the elements Bob actually receives. She also sends  $a = \prod_{i=1}^k a_i$  to Bob through a clear channel.
- 2: Alice picks a random sequence  $R = r_1, \dots, r_{k^2}$ . Alice then sends the rest of  $T(A, R)$  (row  $A_0^R = A$  has already been sent in Step 1) to Bob through the channel. Finally, Alice sends  $A^R$ , the  $k^2$ th row of  $T(A, R)$ , through a clear channel.

**Checking Step:** Bob makes the following checks

- Let  $A^R = a_1^R, \dots, a_k^R$ . Let  $T'(A, R) = [t'_{i,j}]$ , where  $t'_{i,j}$  is the element actually received by Bob (either  $t_{i,j}$  or  $\#$ ). If  $a_i^R \neq t_{k^2,i}$ , for any  $t_{k^2,i} \neq \#$ , then Bob rejects.
- Bob checks all the conditions discussed in the local-verifiability Lemma for randomizing tableaux. If one of them fails, i.e. if Bob sees that the tableau is not a valid randomizing tableau for  $A$ , then Bob rejects.
- Finally, Bob checks that

$$a = \prod_{i=1}^k a_i^R.$$

Figure 3.6: A protocol for proving assertions about products of permutations.

elements. Second, we want to show that, with high probability, only a small amount of very local information is leaked by the protocol.

**Lemma 3.13** [Transfer of Confidence]. Let  $A = a_1, \dots, a_k$ , and  $a$  be as defined in Step 1 of protocol  $\text{PROVE-PRODUCT}(A, k)$ . Suppose that  $a \neq \prod_{i=1}^k a_i$ . Then there exists some constant  $c$  such that no matter what Alice sends in Step 2 of the protocol, Bob will reject with probability  $1/k^c$ .

**Proof:** The proof is virtually the same as with the naive version of the protocol. If Alice sends an invalid tableau for  $A$ , then, due to the local-verifiability lemma, Bob will detect this fact and reject with probability at least  $1/k^{40}$ . If the “ $A^R$ ” sent by Alice is not equal to the top row of the tableau, Bob will detect this fact with probability at least  $1/k^{10}$ .

Before we bound the amount of information transferred by the protocol, we first show that with high probability, there will be  $k - 1$  consecutive rows in the tableau which Bob does not see. This will allow us to use the gap-randomization lemma. To show the existence of  $k - 1$  consecutive unseen rows, we simply show that with high probability, Bob only sees a small number of tableau elements.

**Lemma 3.14** In protocol  $\text{PROVE-PRODUCT}(A, k)$ , the probability that Bob sees more than  $j$  tableau elements, aside from the top row, is bounded above by  $2^{-j}$ .

**Proof:** Using the binomial theorem, we can bound from above the probability that Bob receives exactly  $i$  of the  $k^3$  elements transferred. We have

$$\begin{aligned} \text{prob}(\text{Bob receives } i \text{ elements}) &= \binom{k^3}{i} \left(\frac{1}{k^{10}}\right)^i \left(1 - \frac{1}{k^{10}}\right)^{k^3-i} \\ &\leq \binom{k^3}{i} \left(\frac{1}{k^{10}}\right)^i \\ &\leq \left(\frac{ek^3}{i}\right)^i \left(\frac{1}{k^{10}}\right)^i \\ &\leq \frac{e}{k^{7i}} \end{aligned}$$

Therefore, the probability that at least  $k$  values were received is bounded above by

$$\sum_{i=j}^{\infty} \frac{e}{k^{7i}} \leq \frac{2e}{k^{7j}} \leq 2^{-j}. \quad \blacksquare$$

The existence of a large gap in the tableau follows from this lemma.

**Lemma 3.15** In protocol  $\text{PROVE-PRODUCT}(A, k)$ , the probability that for all  $i$ , Bob receives some element from one of rows  $i + 1, i + 2, \dots, l + k - 2$ , is bounded above by  $2^{-k}$ .

**Proof:** Since there are  $k^2$  rows, discounting the last row, Bob would have to see at least  $k^2/(k - 2) > k$  elements to avoid having a  $k - 2$  row gap. However, by Lemma 3.14, the probability that Bob receives more than  $k$  of these elements is at most  $2^{-k}$ .  $\blacksquare$

We can now show that with high probability, Bob gains extra knowledge, in the information sense, about only a few elements of  $A$ .

**Lemma 3.16** [Security] In protocol  $\text{PROVE-PRODUCT}(A, k)$ , with probability at least  $1 - 2^{-k} - 2^{-j}$ , Bob's view will depend only on the the product  $a = \prod_{i=1}^k a_i$ , and at most  $j$  of the  $a_i$ 's.

**Proof:** With probability at least  $1 - 2^{-k}$ , there will exist an  $i < k^2$  such that rows  $i + 1, \dots, i + k - 2$  are not seen by Bob. By the gap randomness property of tableaus, rows  $i + k - 1, \dots, k^2$  will depend only on  $a$ . With probability at least  $1 - 2^{-j}$ , Bob will see no more than  $j$  elements in rows  $0, \dots, i$ . These elements will belong to at most  $j$  columns,  $i_1, \dots, i_j$ . However, by the columnwise-independence property of tableaus, this view will only depend on elements  $a_{i_0}, \dots, a_{i_j}$ . The probability that no gap exists or that more than  $j$  elements are seen is at most  $2^{-k} + 2^{-j}$ , which implies the lemma.  $\blacksquare$

It should be noted that the above arguments are very insensitive to minor fluctuations in the transferal probabilities. In order to prove the transfer of confidence lemma, it sufficed that each element was received with probability at least  $k^{-c}$  for some  $c$ . In order to prove the security lemma, it sufficed that each element was received with probability at most  $k^{-c}$ , where  $c$  is a sufficiently large constant. Thus, in order to make our lemmas go through, we need only insure the following two properties about the transmission rates.

1. (confidence)  $\text{prob}(\text{Bob receives a group element}) > k^{-c}$ ,  
for some constant  $c$ .

2. (security)  $\text{prob}(\text{Bob receives a group element}) < k^{-c}$ ,  
for  $c$  sufficiently large ( $c > 10$  suffices).

This observation is crucial in the next section, where we replace the group transfer channel with the ordinary oblivious transfer channel.

### 3.3.5 Replacing Group Element Transfer with Oblivious Transfer.

So far, we've shown how to transfer confidence about products of group elements, provided the base group is finite, using a group element oblivious transfer channel. While this channel is useful for expository purposes, we would like our protocols to only rely on the existence of ordinary oblivious transfer. Fortunately, the necessary modifications are very straightforward. We need to address two differences between the channels. First, we need to deal with the fact that we can't actually set the transfer probability. Second, we need to deal with the fact that only bits may be transferred, not arbitrary group elements.

#### Adjusting transfer probabilities.

In the idealized channel, we were able to adjust the transfer probability. For suitable transfer probabilities, we can easily emulate this ability using the following simple trick. If the transfer probability,  $\rho$ , is of the form  $2^{-n}$ , then we can convert each bit,  $b$ , into  $n$  bits,  $b_1, \dots, b_n$ , such that

$$b = b_1 \oplus \dots \oplus b_n.$$

To simulate transferring  $b$  with probability  $2^{-n}$ , we just transfer bits  $b_1, \dots, b_n$ , each with probability  $\frac{1}{2}$ . The receiver can recover  $b$  iff he receives all of the  $b_i$ 's. We call such a  $b$ , implicitly represented as the exclusive-or of a sequence of bits, a "group bit."

#### Transferring bits instead of group elements.

We cannot so easily simulate the ideal transfer of group elements. However, we can approximate this transfer in a manner sufficient for our purposes.

Suppose that the base group,  $G$ , has  $\leq 2^m$  elements. We can uniquely identify each element by  $m$  “group bits”. Each group bit will actually be represented as the exclusive-or of a sequence of bits, as described above. To transfer a group element,  $g$ , we just transfer each of its identifying group bits. If the receiver gets all  $m$  group bits, it can recover  $g$ . Thus, if each group bit is transferred with probability  $> k^{-c}$ , the receiver will get each group element with probability  $k^{-mc}$ . As long as  $m$  is truly a constant, this will not affect the transfer of confidence lemma.

However, if some, but not all of the group bits representing a group element  $g$  are received, the receiver has some partial information about  $g$ . Thus, this simulation is not really faithful. However, if each group bit is transferred with probability  $k^{-c}$ , the probability that any of the bits  $m$  bits are received is bounded above by  $mk^{-c}$ .

When we argue about transfer of confidence, we suffice to assume that the receiver only looks at elements for which it received *all* of the group bits. Such an idealization ignores a lot of information obtained by the receiver, but this is allowable since possessing more information will only increase the receiver’s ability to acquire confidence. Likewise, when we analyze knowledge transfer, we assume that the receiver receives a group element if it received *any* of the group bits in the element’s representation. This is assuming more information on the part of the receiver, but again this is allowable in establishing an upper bound on information obtained.

By setting our simulated bit transfer probability to  $2^{-c\lceil \log k \rceil}$ , for  $c$  sufficiently large, we can insure the following two conditions hold.

1. The receiver receives each element *in toto* with probability  $> k^{-c'}$ , for some constant,  $c'$ .
2. The receiver receives a bit from a given element with probability at most  $mk^{-c} < k^{-10}$ , for  $c$  sufficiently large.

Thus, we can assume an ordinary oblivious transfer channel, without any loss of generality.

## 3.4 Committing Bits With Zero-Knowledge Proofs.

In the previous section, we showed how to prove assertions about products of permutations. In this section, we show how to apply this technique to prove arbitrary  $NP$  assertions about committed bits. To do this, we make use of Barrington's construction for implementing  $NC^1$  circuits in terms of width 5 permutation branching programs (W5PBP's).

Our commital protocol is largely based on the simple-commit protocol. We take the set,  $X$ , of bits produced by simple-committing a set of bits,  $b^1, \dots, b^n$ , and then prove assertions about set  $X$  in a way that only reveals a small number of bits from  $X$ . We can then show that revealing this small number of bits does not compromise the security of the original simple-commit protocol.

### 3.4.1 Bounding the complexity of the predicates we must prove.

While as an end result, we would like to prove arbitrary  $NP$  assertions with our formalism, our task would be much easier if we only had to prove very simple assertions. In fact, it suffices to be able to prove  $NC^1$  assertions. By a well-known result of complexity theory, any  $NP$  predicate,  $\mathcal{P}(x_1, \dots, x_n)$ , may be replaced by an existential quantifier, and an  $NC^1$  predicate. Thus, there exists a constant  $c$ , and an  $NC^1$  predicate  $P$ , such that

$$\mathcal{P}(x_1, \dots, x_n) \iff P(x_1, \dots, x_n, y_1, \dots, y_{n^c}).$$

In order to commit a set of bits,  $x_1, \dots, x_n$ , and prove some  $NP$  assertion  $\mathcal{P}$  about them, one simply determines the appropriate witness bits,  $y_1, \dots, y_{n^c}$ , commit them along with the  $x$ 's, and prove the corresponding assertion,  $P$ . The *witness* bits,  $y_1, \dots, y_{n^c}$ , may be determined in polynomial time, given an accepting run of a nondeterministic polynomial-time machine which computes  $\mathcal{P}$ . Thus, this conversion is without any significant computational overhead. Paradoxically, it helps to commit bits which one has no intention of decommitting.

### 3.4.2 Preliminaries

In the SIMPLE-COMMIT protocol, Alice commits a bit,  $b$ , to Bob by writing it as an exclusive-or of several bits,  $b_1, \dots, b_k$ , and then transferring these bits to Bob using an oblivious transfer channel. If Alice simple-commits a set of bits,  $b^1, \dots, b^n$ , she writes  $b^i = b_1^i \oplus \dots \oplus b_k^i$ , for  $1 \leq i \leq n$ . For notational convenience, we designate this set of bits by  $X = x_1, \dots, x_{nk}$ , where  $x_{k(i-1)+j} = b_j^i$ . At the end of the SIMPLE-COMMIT protocol, bit  $x_i$  is known to Bob with probability  $1/2$ , independently from all the other bits of  $X$ .

**Converting predicates on  $b^1, \dots, b^n$  to predicates on  $X$ .**

Given all the bits of  $X$ , one can trivially reconstruct the bits  $b^1, \dots, b^n$ . This gives a natural of converting a predicate  $P(b^1, \dots, b^n)$  into an equivalent predicate,  $P_x(X)$ . We say that  $P_x(X)$  is true iff  $P(b^1, \dots, b^n)$  is true, where

$$b^i = \bigoplus_{j=1}^k x_{k(i-1)+j}.$$

Essentially, all we've done is to account for the trivial encoding of our bit-sequence.

Since our decoding procedure is an easy  $NC^1$  operation, the complexity of predicate  $P_x$  will be about the same as the complexity of  $P$ . In particular, if  $P$  is in  $NC^1$ , then  $P_x$  will also be in  $NC^1$ .

### 3.4.3 Evaluating $NC^1$ Predicates as Products of Permutations

Given some predicate,  $P(x_1, \dots, x_{kn})$ , Barrington [B] shows how to convert it into a width-5 permutation branching program,  $B_P$  of length  $m = (kn)^c$ , for some constant  $c$  based on the complexity of  $P$ . We can write this branching program as

$$B_P = ((i_1, \pi_1^0, \pi_1^1), (i_2, \pi_2^0, \pi_2^1), \dots, (i_m, \pi_m^0, \pi_m^1), a),$$

where for  $j \in [1, m]$ , and  $k \in \{0, 1\}$ , we have  $i_j \in [1, nk]$ , and  $\pi_j^k \in S_5$ . Element  $a$ , the “accept” element, is in  $S_5 - I$ , where  $I$  is the identity element.

The group  $S_5$  is the group of permutations on 5 elements. To evaluate a branching program,  $B_P$ , on input  $(x_1, \dots, x_{nk})$  we simply compute

$$B_P(x_1, \dots, x_{nk}) = \prod_{j=1}^m \pi_j^{x_{i_j}}.$$

The result of this computation is some element in  $S_5$ . By Barrington's construction, we have

$$B_P(x_1, \dots, x_{nk}) = \begin{cases} a & \text{if } P(x_1, \dots, x_{nk}) \\ I & \text{otherwise.} \end{cases}.$$

Formally, we needn't include  $a$  as part of the definition; we can say that a branching program accepts iff the resulting product is not the identity. However, for our purposes, we need the fact that revealing the product of the permutations gives away no more information than revealing whether the predicate is true.

#### 3.4.4 The Zero-Knowledge Bit-Comital Protocol.

Since we can express the truth of an  $NC^1$  predicate in terms of evaluating the product of a sequence of permutations, we can use the machinery of the last section to give a simple bit-commital protocol. As usual, we adopt the convention that Alice possesses some bits, which she wishes to commit to Bob, along with a proof of some  $NC^1$  predicate,  $P$ .

**Remark:** The group element transfer procedure is chosen to have the following properties. For any element  $g$  which is represented in binary as  $g_1 \dots g_7$ , Bob can reconstruct each bit  $g_i$  with probability less than  $k^{-10}/7$ . Hence,

$$\text{prob}(\text{Bob gets information about } g) < k^{-10}.$$

Similarly,

$$\text{prob}(\text{Bob gets complete information about } g) > k^{-71},$$

for  $k$  sufficiently large.

Whereas in the commital protocol, one may wish to commit a set of bits, in the decommital protocol, one may only wish to decommit a single bit. This is easily accomplished using essentially the same protocol as SIMPLE-DECOMMIT.

**Protocol Zero-Knowledge-Commit( $b^1, \dots, b^n, k, P$ )**

- 1: Alice chooses a random  $X = x_1, \dots, x_{nk}$ , subject to

$$b^i = \bigoplus_{j=1}^k x_{k(i-1)+j}.$$

Alice transfers  $X$  to Bob using the oblivious transfer channel. We denote by  $X' = x'_1, \dots, x'_m$  the set of values which Bob receives.

- 2: Using some canonical procedure, Alice computes

$$B_P = ((i_1, \pi_1^0, \pi_1^1), (i_2, \pi_2^0, \pi_2^1), \dots, (i_m, \pi_m^0, \pi_m^1), a),$$

a branching program that computes  $P_x$ . We assume without loss of generality that  $m \geq k$ , since branching programs may be trivially padded by including triples of the form  $(i_j, I, I)$ . Alice computes  $A = a_1, \dots, a_m$ , by

$$a_j = \pi_j^{x_{i_j}}.$$

- 3: Alice executes  $\text{PROVE-PRODUCT}(A, m)$ , using an ordinary oblivious transfer channel. Specifically, each group elements  $g \in S_5$  to be transferred is broken up into 7 bits,  $g_1, \dots, g_7$  (the 7 comes from the fact that  $|S_5| = 120$ ). Each bit,  $g_i$ , is broken up into  $l = 10\lceil \log m \rceil + 3$  bits, which we write,

$$g_i = g_i^1, \dots, g_i^l.$$

Each bit,  $g_i^j$  is then transferred with probability  $1/2$ . Finally, the value of  $X$  is saved for later use.

**Checking Step:** Bob makes the checks given in Figure 3.8. He aborts if any of the checks fails.

Figure 3.7: A protocol for bit commital with zero-knowledge proofs.

**Checks Bob makes in protocol ZERO-KNOWLEDGE-COMMIT.**

- Bob makes all the checks from protocol PROVE-PRODUCT. For checking purposes, Bob treats any element whose bits were not all received as '#', i.e. as unreceived.
- Whenever he receives bit  $x_l$ , and whenever he completely receives element  $a_j$ , where  $i_j = l$ , Bob checks that

$$a_i = \pi_j^{x_{ij}}.$$

Since  $B_P$  is canonical, Bob can indeed make this check.

- Finally, Bob checks that the product of the  $a_i$ 's, as asserted by Alice, is equal to the "accept" permutation,  $a$ , of  $B_P$ .

Figure 3.8: Consistency checks for protocol ZERO-KNOWLEDGE-COMMIT.

**Protocol Zero-Knowledge-Decommit( $X, i, n, k$ ) /\* decommit bit  $b^i$ , \*/**

Alice sends Bob the values of  $x_{k(i-1)+j}$ , for  $j \in [1, k]$ , over a clear channel. Bob checks the validity of his decommital by comparing these bits to  $X'$ . If, for any value of  $i, j$ ,  $x'_{k(i-1)+j} \neq \#$ , and  $x'_{k(i-1)+j} \neq x_{k(i-1)+j}$ , then Bob rejects. Otherwise, he computes

$$b^i = \bigoplus_{j=1}^k x_{k(i-1)+j}.$$

Figure 3.9: Decommital protocol for zero-knowledge bit commital scheme.

### 3.4.5 Properties of the Zero-Knowledge-Comital Protocol.

We wish to establish four properties of our protocols. These are,

1. (Reliability) Alice cannot trick Bob into recovering a different bit than was committed without Bob detecting this with nonnegligible probability.
2. (Confidence) If  $P(b^1, \dots, b^n)$  is false, then Bob will reject with nonnegligible probability.
3. (Independence) Decommitting one bit gives no information about any other bits, other than what can be inferred by knowing the value of the committed bit.
4. (Security) After ZERO-KNOWLEDGE-COMMIT, with high probability, Bob gains no information, in an information theoretic sense, about the values  $b^1, \dots, b^n$ .

The properties of reliability and confidence are relatively straightforward to prove. The properties of independence and security require some more machinery, which we will develop in the next section.

Before giving the proofs of reliability and confidence, we first note that while a malicious Alice may violate the comital and decommital protocols as much as she wishes, it still makes sense to refer to her committed bits. We define the committed bits,  $b^1, \dots, b^n$ , by

$$b^i = \bigoplus_{j=1}^k x_{k(i-1)+j}.$$

While she may pick whatever value of  $X$  she wishes to send, she still must send it.

**Lemma 3.17** [Reliability] For any strategy Alice may employ, the probability that, at the end of protocol ZERO-KNOWLEDGE-DECOMMIT, Bob reconstructs some bit  $b^{i'} \neq b^i$  is at most  $\frac{1}{2}$ .

**Proof:** This property carries over from the base protocols, SIMPLE-COMMIT, and SIMPLE-DECOMMIT, and the proof is identical. ■

**Lemma 3.18** [Confidence] There exists some  $c$  such that if  $P(b^1, \dots, b^n)$  is false, then Bob will reject after ZERO-KNOWLEDGE-COMMIT with probability at least  $k^{-c}$ , for  $k$  sufficiently large.

**Proof:** There are three possibilities we must consider.

1. Alice behaves honestly, picking both the correct sequence  $A$ , and following protocol PROVE-PRODUCT.
2. Alice picks  $A$  correctly, but violates the protocol PROVE-PRODUCT.
3. Alice picks  $A$  incorrectly.

In Case 1, Alice will convey the correct value of the branching program, and thus the predicate, on  $X$ . In this case, Bob will certainly reject if the predicate is false. In Case 2, by Lemma 3.13, Bob is guaranteed to reject with probability at least  $m^{-c'}$ , for some  $c'$ . Since  $m \leq k^{c''}$ , for some  $c''$ , we can pick  $c = c'c''$ . Finally, in Case 3, Bob will detect an incorrect element of  $A$  if he sees the entire element, and the bit it depends on. This will happen with probability at least  $k^{-71}/2$ , when  $k$  is sufficiently large. ■

## 3.5 Simulation Results for Zero-Knowledge Bit Committal.

### 3.5.1 Introduction.

In order to incorporate our bit-committal protocols into larger protocols, it is necessary to fit them into the simulation paradigm. Thus, in order to bound how much information the receiver obtains at various times during the committal and decommittal protocols, we construct a simulator which approximates the view seen by the verifier. We require that this approximation be statistically close; it must be impossible to distinguish the simulated distribution from the actual one, with probability greater than  $\frac{1}{2} + 1/n^c$ , for any  $c$ , given only a polynomial number of samples from each distribution.

### Why is Simulation Possible?

On the face of it, simulation results are paradoxical. Generally, simulators are required to run in probabilistic polynomial time. However, they must

simulate views obtained from arbitrarily powerful entities. For instance, we might ask a simulator to simulate the commital of a sequence of bits,  $b^1, \dots, b^n$ , along with a proof that they encode a Hamiltonian cycle of some arbitrary graph. If the simulator actually ran the zero-knowledge commital protocol, it would have to find a Hamiltonian cycle, or be caught some nonnegligible fraction of the time.

The reason it is possible for a weak simulator to simulate powerful entities is because it does *not* have to use an actual oblivious transfer channel. It's sole requirement is to reproduce what the receiver would hear at the receiving end of the channel. Thus, the simulator can randomly decide which bits will get through the simulated channel, and then decide what values these bits will actually have. This is in sharp contrast to the actual oblivious transfer channel, where one decides what values the bits in ones message will have, and then the channel randomly decides which bits will get through. This extra flexibility makes our simulation results possible.

## What must be simulated?

The commital protocol leaks information in three separate ways, each of which must be dealt with. First, the commitor sends  $X$  through an oblivious transfer channel. Second, she sends a randomized tableau based on  $X$  through an oblivious transfer channel. Third, she sends the top row of the tableau in the clear. The first way of leaking knowledge is easily analyzed; the bulk of this section will be involved with formally analyzing knowledge leaked by revealing portions of the tableau.

### 3.5.2 The Notion of Masks.

Given a sequence of  $n$  bits,  $B = b_1, \dots, b_n$ , that are sent through an oblivious transfer channel, we can model the action of the channel as follows. The channel first generates a random sequence of  $n$  bits,  $F = f_1, \dots, f_n$ . It then outputs  $B' = b'_1, \dots, b'_n$ , where  $b'_i = b_i$  if  $f_i = 1$ , and  $b'_i = \#$  otherwise. We call  $F$  a *mask* for the channel, and adopt the notation,

$$B' = B \oplus F.$$

Also, if we are considering some subset of  $B$ , it makes sense to view  $F$  as masking this subset as well. Thus, we can speak of

$$\{b_1, b_5, b_7\} \oplus F.$$

If,

$$F = [1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1]$$

, this would yield  $\{b_1, \#, b_7\}$ .

A simple but important feature of masks is that they are independent of the data actually being transmitted. Also, it is easy to generate masks with the correct distribution; it suffices to flip  $n$  coins. The separation of masks and the data they mask is useful for our simulator construction. Our simulator first generates a mask with the correct distribution, then simulates the masked data.

### 3.5.3 A simple example.

Suppose someone generates the following distribution. He chooses the bit sequence  $b_1, \dots, b_n$ , uniformly, subject to the constraint that

$$\bigoplus_{i=1}^n b_i = 1 \iff \text{God exists.}$$

It seems difficult to statistically approximate this distribution. If one guesses wrong, an infinitely powerful being can detect ones error after seeing one sample of the simulated distribution. However, one *can* simulate the distribution  $B \oplus F$ , where  $F$  is a uniformly distributed mask. One simply generates  $B$  and  $F$  uniformly, and output  $B \oplus F$ . If  $F$  consists entirely of 1's, then  $B \oplus F = B$ , and one will be detected (unless one is enlightened, or lucky). In all other cases, it is impossible for even an infinitely powerful being to distinguish the simulated distribution from the actual one. Since,  $F$  consists entirely of 1's only  $2^{-n}$  of the time, the two distributions are indistinguishable.

### 3.5.4 Preliminary Conventions.

Our simulator construction is complicated by the complexity of the data structures it must work with. It must simulate, among other things, a masked

tableau, which is an array of elements of  $S_5$ , which are represented by a sequence of 7 group bits, each of which is broken up into an exclusive-or of a sequence of bits. Fortunately, this structure is hierarchical, and we can organize the simulator along these lines. To facilitate this approach, we introduce the following terminology for talking about how much a mask reveals about higher level features.

**Definition 3.7** Given a mask,  $F$ , which is to be applied to a tableau,  $T$ , we say that  $F$  *exposes* a bit,  $b$ , of  $T$ , if it is equal to 1 on the location corresponding  $b$ 's location in the tableau. In other words,  $F$  exposes a bit if this bit would still be seen when  $T$  is masked by  $F$ . We say that  $F$  exposes a group bit,  $b$ , if it exposes *all* the bits that make  $b$  up. We say that  $F$  exposes a group element,  $g$ , if it exposes *any* of the bits that make  $g$  up. We say that  $F$  exposes a column of  $T$  if it exposes *any* element in that column. We say that  $F$  exposes a row of  $T$  if it exposes *any* element in that row.

In the above definition, it is important to distinguish a bit from its value. When we talk about exposing a particular bit, or other feature, we are actually referring to a tableau position. Whether a mask exposes or fails to expose a feature depends on the mask and the position of the feature in the tableau, but not on any of the actual values of the tableau elements.

The above definitions of when  $F$  exposes something make sense for any tableau, regardless of how the base row was actually generated. In the case where the tableau is generated from a branching program, we can also formulate what it means for  $F$  to expose a branching program variable.

**Definition 3.8** Let

$$B = ((i_1, \pi_1^0, \pi_1^1), (i_2, \pi_2^0, \pi_2^1), \dots, (i_m, \pi_m^0, \pi_m^1), a)$$

be a branching program on variables  $x_1, \dots, x_n$ . Let  $A = a_1, \dots, a_m$  be defined by

$$a_i = \pi_j^{x_{i_j}}.$$

Let  $T$  be a tableau with base row  $A$ . We say that a mask  $F$  exposes variable  $x_i$  if, for some  $j$  such that  $i_j = i$ ,  $F$  exposes column  $j$  of  $T$ .

### 3.5.5 Simulating Unexposed Tableau Features.

It turns out to be very simple to simulate the masked values of unexposed tableau features. An unexposed bit,  $b$ , will always be turned into the special symbol, “#”, when it is masked by  $F$ , by definition of bit exposure. Thus, when simulating  $T \oplus F$ , the simulator simply writes a # in this position, without even hazarding a guess about what the bit’s value may have been.

Simulating the masked value of an unexposed group bit,  $g$ , is somewhat more subtle, since some of the bits of  $g$  may be exposed. However, we can make the same argument as with the toy example described in the introduction. Recall that  $g$  is written as  $g_1 \oplus \dots \oplus g_l$ , where  $g_1, \dots, g_l$  are chosen uniformly subject to this constraint on their parity. Suppose that bit  $g_i$  is unexposed. If one doesn’t know  $g_i$ , then one has no information about  $g$ , and the values of the other bits are distributed uniformly. Thus, the simulator can pick the values of  $g_1, \dots, g_l$  uniformly, and then compute the result of masking by  $F$ .

As soon as one can simulate the masked values of unexposed group bits, one can simulate all the masked values of unexposed higher level features. For instance, one simulates the masked value of an unexposed group element by simulating the masked values of each of its 7 group bits. But all of these group bits are by definition unexposed, and thus may be handled as described above.

### 3.5.6 Bounding the number of exposed tableau elements.

We now argue that, with high probability, a random mask,  $F$ , exposes only a few tableau elements of  $T$ . To do this, we first show that a random  $F$  exposes each element of a tableau with sufficiently low probability.

**Lemma 3.19** Let  $T$  be a randomizing tableau, where each group bit is broken up into  $10\lceil \log m \rceil + 3$  bits. Let  $g$  be a group element of  $T$ , and let  $F$  be a uniformly chosen mask. Then,

$$\text{prob}(g \text{ is exposed by } F) < \frac{1}{m^{-10}}.$$

**Proof:** Let  $g = g_1, \dots, g_7$ . where  $g_i$  is a group bit. Group bit  $g_i$  is exposed iff all of its bits are exposed. Since each bit is exposed with probability  $\frac{1}{2}$ ,

we have

$$\begin{aligned} \text{prob}(g_i \text{ is exposed by } F) &= \left(\frac{1}{2}\right)^{10\lceil\log m\rceil+3} \\ &\leq \frac{m^{-10}}{8}. \end{aligned}$$

Now, since  $g$  is exposed only if one of its bits is exposed, we have

$$\begin{aligned} \text{prob}(g \text{ is exposed by } F) &\leq 8 \cdot \text{prob}(g_i \text{ is exposed by } F) \\ &\leq 7 \left(\frac{m^{-10}}{8}\right) \\ &< \frac{1}{m^{10}} \quad \blacksquare \end{aligned}$$

Having bounded the probability of a single element being exposed, it is easy to show that, with high probability, only a few elements will be exposed.

**Lemma 3.20** Let  $T$  be an  $m^2$ -stage randomizing tableau, whose rows are of length  $m$ , and where each group bit is broken up into  $10\lceil\log m\rceil + 3$  bits. Let  $F$  be a uniformly chosen mask. Then, with probability at least  $1 - 2^{-j}$ , only  $j$  elements not in the top row will be exposed by  $F$ .

**Proof:** By Lemma 3.19, each element of the tableau will be exposed with probability at most  $m^{-10}$ . The argument then proceeds exactly as with the proof of Lemma 3.14.  $\blacksquare$

**Remark:** We exclude the top row from our argument for two reasons. First, this allows us to use the same calculations as with the proof of Lemma 3.14. Second, in the protocol we wish to analyze, the top row of the tableau is revealed in the clear.

Lemma 3.20 motivates the following definition.

**Definition 3.9** Let  $T$  be an  $m^2$ -stage randomizing tableau, whose rows are of length  $m$ , and where each group bit is broken up into  $10\lceil\log m\rceil + 3$  bits. We say a mask,  $F$ , is  $c$ -tame if it exposes at most  $m^c$  elements of  $T$ .

Using our lemma, we have that if  $F$  is chosen at random,

$$\text{prob}(F \text{ is } c\text{-tame}) \geq 1 - 2^{-m^c}.$$

Thus, for the purposes of our simulation, we may safely assume that, for any constant  $c > 0$ , a uniformly chosen  $F$  will be  $c$ -tame.

### 3.5.7 Simulating $(T(A, R) \oplus F, A^R)$ by making queries about $A$ .

Using some of the machinery outlined in the previous sections, we can begin to derive some crude simulation results. These results will not be ideal - they will not give the desired bounds on the amount of information transferred - but will be good enough to be used in the proof of our main lemmas.

Suppose we have a mask  $F$ , which is  $c$ -tame for some constant  $c < 1$ . Suppose some sender has an unknown permutation sequence  $A = a_1, \dots, a_m$ , where  $a_i \in S_5$ , and  $a = \prod_{i=1}^m a_i$  is known. This sender chooses an  $m^2$ -stage tableau,  $T = T(A, R)$  for some uniformly chosen  $R$ . He uses the standard conventions (i.e. those used in the bit-committal protocol) for representing group elements and group bits. Then he sends, to some receiver,  $T \oplus F$ , and  $A^R(T)$ , the top row of  $T$ . We wish to simulate the distribution  $(T \oplus F, A^R)$ , using as little information about  $A$  (other than its product) as possible.

Clearly, if we have no information about  $A$ , other than its product, we cannot succeed. A mask,  $F$ , even if it is  $c$ -tame, may expose the tableau position on the bottom row corresponding to  $a_5$ . In this case, one would have to know some information about  $a_5$  in order to carry out the simulation. We therefore adopt a less arduous requirement for our simulator. We allow them to ask for particular values of  $A$ . For instance, if  $F$  exposed  $a_5$ , the simulator can choose to ask for the value of  $a_5$ .

Given a base row, creating a random tableau is easy, so our simulator can easily carry out its task given access to all of  $A$ . However, for our purposes, we wish to bound the number of queries our simulator makes.

**Lemma 3.21** [The Tableau Simulation Lemma] Let  $A = a_1, \dots, a_m$ , where  $a_i \in S_5$ , and  $a = \prod_{i=1}^m a_i$ . Let  $F$  be a  $c$ -tame mask, where  $c < 1$ . Let  $T_m$  be the uniform distribution over  $m^2$ -stage tableaus,  $T(A, R)$ , and let  $T_m$  be represented as with the committal protocol. Consider the distribution  $\mathcal{T}_{A,m} = (T_m \oplus F, A^R(T_m))$ , where  $A^R(T_m)$  is the topmost row of  $T_m$ . There exists a simulator,  $M(m, a, F)$ , with the following properties.

1. On input  $a, m, F$ , where  $m$  is written in unary,  $M(a, m)$  runs in expected polynomial time.
2.  $M(a, m)$  realizes the distribution  $\mathcal{T}_{A,m}$ .
3.  $M(a, m)$  makes at most  $m^c$  queries about  $A$ .

**Proof:** Our proof is by construction. We give our simulator in Figure 3.10. First, we make some simple observations about  $F$ . For convenience, let  $l = m^c$ . Mask  $F$  exposes at most  $l$  rows and  $l$  columns of  $T_m$ . This follows from the fact that  $F$  exposes only  $l$  elements of  $T_m$ . We denote by  $I = \{i_1, \dots, i_l\}$  the column numbers exposed by  $F$ . Since only  $l < m$  rows are exposed, there must exist some  $j$  such that rows  $j+1, \dots, j+m-2$  are all unexposed, by a trivial counting argument. Using these facts, we now construct the simulator.

The simulator described in Figure 3.10 clearly works in expected polynomial time. It only asks for at most  $l = m^c$  values of  $A$ . It suffices then to show that the simulator approximates the correct distribution. In fact, we can show that it realizes exactly the distribution  $(T \oplus F, A^R)$ . It is necessary to establish two facts.

1. The partial tableau produced in steps 4.1-4.5 is correctly distributed. In other words, the partial tableau construction produces the same distribution over its defined region as would be obtained if one were to sample the actual tableau distribution, and project these samples into the same defined region.
2. Filling in the undefined regions with random bits gives the same distribution, after masking by  $F$ , as would be obtained if these regions were filled in correctly.

### Analyzing the partial tableau distribution.

First, we note that the decomposition of group elements to bits in Step 4.5 is identical to the decomposition used in the bit-committal protocol. Therefore, we can look at the distribution of partial tableaus on the group element level. If the distribution of group elements is correct, so will the distribution on the bit level. We now must show that Steps 4.1, 4.2, 4.3 and 4.4 correctly fill in their regions of the partial tableau.

Step 4.1 simply fills in the relevant entries of the bottom row with the values of their corresponding entries in  $A$ . Since the bottom row of the actual tableaus are always equal to  $A$ , Step 4.1 clearly realizes the correct distribution on its domain.

Step 4.2 produces the correct distribution on its domain, due to the columnwise independence lemma. In the actual construction of a tableau with a

**Simulator Masked-Tableau(a,m,F)**

- 1: Compute  $I, j$ , as defined above. Ask for the values of  $a_{i_1}, \dots, a_{i_l}$ , for  $i_k \in I$ .
- 2: Uniformly choose  $R_{lower} = r_1, \dots, r_j$ , where  $r_i \in S_5$ . Likewise, uniformly choose  $R_{upper} = r_{j+m}, \dots, r_{m^2}$ . The sequences  $R_{lower}, R_{upper}$  will be used as partial randomizing sequences, used to create portions of a tableau.
- 3: Choose  $A' = a'_1, \dots, a'_m$ , uniformly, subject to

$$a = \prod_{i=1}^m a'_i.$$

- 4: Construct a partial tableau  $T = [t_{f,g}]$ , as shown in Figure 3.11.
- 5: The tableau  $T$  will have many of its group elements undefined. Consequently, the group bits which make up these elements, and the bits which make up these group bits, will also be undefined. Uniformly select the values of the undefined bits. We make no restrictions on the group elements implicitly defined by this process, nor even require that these elements are well-defined.
- 6: Output  $T \oplus F, A^R(T)$ . Note that the top row of  $T$  will be well-defined, since all rows above  $j + m - 1$  are completely defined.

Figure 3.10: Simulating a masked tableau.

**Creating a masked Tableau.**

- 4.1:** For  $g \in I$ , compute  $t_{0,g} = a_g$ .
- 4.2:** For  $f \in [1, j], g \in I$ , compute  $t_{f,g}$ , using the construction specified in the definition of randomizing tableaux, and the partial randomizing sequence,  $R_{lower}$ . Note that in the definition, each tableau element is expressed as a function of the randomizing sequence and the element just below it in the tableau. Hence, this construction is well defined.
- 4.3:** For  $g \in [1, m]$ , compute  $t_{j+m-1,g} = a'_g$ .
- 4.4:** For  $f \in [j+m, m^2], g \in [1, m]$  compute  $t_{f,g}$  using the construction specified in the definition of randomizing tableaux, and the partial randomizing sequence,  $R_{upper}$ .
- 4.5:** Decompose each defined element  $t_{f,g}$  into its group bits. Randomly decompose each group bit,  $b$ , into  $q = 10\lceil \log m \rceil + 3$  bits,  $b_1, \dots, b_q$  such that

$$b = \bigoplus_{i=1}^q b_i.$$

Figure 3.11: Creating a partial tableau.

given base row,  $A$ , using a randomizer  $R = r_1, \dots, r_m$ , we can compute  $t_{f,g}$ , by

$$t_{f,g} = F_f(t_{0,g}, r_1, \dots, r_f),$$

where  $F_f$  is a function depending only on  $f$  and, implicitly,  $m$ . Step 4.2 uses the same functions to compute its elements as in the actual construction. It uses the same distribution on the randomizing elements, and starts out with exactly the same base elements (in its domain).

To prove that Step 4.3 fills out row  $j + m - 1$  with the correct conditional distribution, given the entries already filled in, we invoke the gap-randomization lemma. By this lemma, the conditional distribution of row  $j + m - 1$ , given rows  $0, \dots, j$ , will be uniform over all possible rows with the correct row-product. Row  $j + m - 1$ , which is equal to  $A'$ , is clearly given this uniform distribution, by Step 3 of the simulator.

The analysis of Step 4.4 is identical to the analysis of Step 4.2. Instead of building up from the base row, we simply build up from row  $j + m - 1$ . To wit, for  $f > j + m - 1$ , we can write

$$t_{f,g} = G_{j+m-1,f}(t_{j+m-1,g}, r_{j+m}, \dots, r_f),$$

where  $G_{j+m-1,f}$  is a function depending only on  $j, m$ , and  $f$ .

As a trivial consequence of the above argument, masking the simulated partial tableau with  $F$  yields the same distribution as masking the corresponding portions of the actual tableau with  $F$ . Masking only destroys information, making distributions less distinguishable from each other. Furthermore, the top row of the tableau is properly distributed, since it is part of the partial tableau.

### Filling in the tableau with random bits.

Having established that the output of the simulator is valid for the top row, and portion of the tableau which contains all the exposed group bits, we now only have to establish that it outputs a correct distribution on all the unexposed portions of the tableau. However, the masked version of this region is just the union of a set of masked group bits. By the argument in the previous section, one might as well pick their bit vectors at random. If a mask  $F$  does not expose a group bit, then it masks one of the bits of the bit vector representing the group bit. If one projects a set of variables

onto a proper subset, it doesn't matter whether these vectors are uniformly distributed, or uniformly distributes subject to a parity constraint. The resulting distribution is the same. In other words, if you don't see all the variables, you can't make even an educated guess as to their parity.

Thus, we have established that the output of the simulator yields a correct distribution on the top row of the tableau, and on all portions of the masked tableau. The lemma follows. ■

### 3.5.8 Simulating the Committal/Decommital Protocols.

In the previous section, we were able to simulate masked tableaus, along with their top rows. Such a simulation looks promising, since the heart of the commital protocol involves giving the verifier a masked tableau. However, the simulation had two drawbacks. It required that the mask be sufficiently regular, and it required the simulator to make a small number of queries about the base row of the tableau. The first problem is not serious, since, by Lemma 3.20, a randomly chosen mask will be sufficiently regular with probability extremely close to 1. However, the second problem is more challenging to deal with. We get around it by simulating the answers to the queries asked by the tableau simulator.

#### Converting queries about $A$ into queries about $X$ .

Our simulator for masked tableaus worked regardless of how the base row,  $A$  was initially created. However, in the commital protocol, Alice computed  $A$  from a branching program,  $B_P$ , and a set of input variables,  $X = x_1, \dots, x_{nk}$ . The branching program is known to everyone, including any simulator we construct. The value of  $X$ , on the other hand, contains enough information to reconstruct the committed bits, and therefore is not known to the simulator. However, we can directly convert queries about  $A$  into queries about  $X$ . This is accomplished by using the identity,

$$a_j = \pi_j^{x_{i_j}},$$

where  $i_j, \pi_j^0, \pi_j^1$  are determined by  $B_P$ . Note that queries about two different elements of  $A$  may sometimes be answered by a single query about  $X$ . In

any case, the number of queries about  $X$  required to run this portion of the simulation is bounded above by the number of queries that were originally made of  $A$ .

**A simulator for the commital protocol which makes queries of  $X$ .**

We now exhibit a partial simulator for the commital protocol. This simulator assumes that someone actually creates the bit sequence,  $X$ . During the course of the simulation, we allow this simulator to make queries about  $X$ . Note that such a simulation result in no way bounds any knowledge transfer, since having even limited access to  $X$  *a priori* gives one a great deal of information about the committed bits. In the next section, we will show how to simulate these queries, which will complete this particular simulation project. We assume that  $k \geq n^c$ , for some constant  $c$ .

The following lemma states that the simulator given in Figure 3.12 will, if given access to the proper distribution of  $X$ , output a sufficiently close approximation to the output of the actual ZERO-KNOWLEDGE-COMMIT protocol.

**Lemma 3.22** Let  $b^1, \dots, b^n$  be a set of bits, and let  $X = x_1, \dots, x_{nk}$  be distributed uniformly, subject to

$$b^i = \bigoplus_{j=1}^k x_{k(i-1)+j}.$$

Then,  
the induced distribution of simulator PARTIAL-ZERO-KNOWLEDGE( $n, k, P$ ) will be indistinguishable from the output of protocol ZERO-KNOWLEDGE-COMMIT( $b^1, \dots, b^n, k, P$ ), for  $k$  sufficiently large.

**Proof:** The output of protocol ZERO-KNOWLEDGE-COMMIT may be broken into two components:

1. The transmission of  $X$ , chosen according to the distribution stated in the lemma.
2. The transmission of an appropriately chosen randomizing tableau,  $T$ , with base row  $A$  (a function of  $X$  and  $P$ ).

**Simulator Partial-Zero-Knowledge( $n, k, P$ )**

1: Compute

$$B_P = ((i_1, \pi_1^0, \pi_1^1), \dots, (i_m, \pi_m^0, \pi_m^1), a),$$

the branching program for  $P_x$  on input  $X$ . There exists some easily computable constant,  $c > 0$  such that  $k > m^{2c}$ . We also assume that  $m \geq nk$ , a condition which may be easily enforced by “padding” the W5PBP’s with identity permutations.

2: Uniformly choose a mask,  $F_X$ , of length  $nk$ . Mask  $F_X$  will be used to simulate the transfer of  $X$ . Uniformly choose a mask,  $F_T$ , of length  $q = 7M(M^2 + 1)(10\lceil \log m \rceil + 3)$ , which will be used to simulate the transfer of the tableau. Notationally, we write  $F_X = fx_1, \dots, fx_{nk}$ , and  $F_T = ft_1, \dots, ft_q$ . If  $F_T$  isn’t  $c$ -tame, then abort the simulation.<sup>a</sup>

3: For all  $i$  such that  $fx_i = 1$ , query the value of  $x_i$ . Using these answers, compute  $X \oplus F_X$ .

4: Run simulator MASKED-TABLEAU( $a, m, F_T$ ). Whenever MASKED-TABLEAU asks for the value of  $a_j$ , ask for the value of  $x_j$  (unless it has already been asked for), and send the value of  $\pi_j^{x_j}$  back to MASKED-TABLEAU.

5: Simulator MASKED-TABLEAU will output something of the form,  $(T \oplus F_T, A^R(T))$ . Output  $(X \oplus F_X, T \oplus F_T, A^R(T))$ .

Figure 3.12: Partial component of zero-knowledge bit-committal simulator.

---

<sup>a</sup>Note that the simulator will be aborted with only negligible probability.

We handle these two components separately. First, the transmission of  $X$  is equivalent to generating a uniformly distributed tableau,  $F_X$ , and outputting  $X \oplus F_X$ . This is precisely the distribution of the first component output by PARTIAL-ZERO-KNOWLEDGE, provided it doesn't abort. Note that the distribution of  $X$  stated in the lemma is equal to the distribution of  $X$  used by ZERO-KNOWLEDGE-COMMIT. The transmission of the tableau,  $T$ , with base row  $A$ , is equivalent to the following steps.

1. Uniformly choose mask  $F_T$ . The protocol, of course, transmits the tableau, rather than masking it. However, the transmission of a tableau, or any other data structure, implicitly defines a mask, which is equal to 1 on all the bit locations that are actually received.
2. Compute  $A$ , based on  $X$  and  $P$ .
3. Produce, with a uniformly distributed randomizing sequence,  $R$ , a tableau  $T = T(A, R)$ , and output  $T \oplus F_T$ .

Now, PARTIAL-ZERO-KNOWLEDGE also uniformly chooses its mask,  $F_T$ , and computes  $A$  according to the same algorithm. Simulator MASKED-TABLEAU will perfectly realize the output  $T(A, R) \oplus F_T$ , provided that  $F_T$  is  $c$ -tame, for some constant  $c < 1$ . Thus, if there were no tameness restrictions on  $F_T$ , we would be done. In other words, the distribution produced by the simulator, conditioned on  $F_T$  being  $c$ -tame, is equal to the distribution of ZERO-KNOWLEDGE-COMMIT, conditioned on the mask implicitly defined by the transmission of  $T$  being  $c$ -tame. Furthermore, the probability that  $F_T$  isn't  $c$ -tame is the same as for the (implicitly defined) mask used in ZERO-KNOWLEDGE-COMMIT. Thus, for a large set of possible outputs,  $O = (X \oplus F_X, T \oplus F_T)$ , the probability that  $O$  is output by the simulator is equal to the probability that  $O$  is output by the actual commital protocol.

To complete the proof, we now argue that this set of exactly simulated outputs occurs with probability very close to 1. This is equivalent to bounding above the probability that a uniformly chosen  $F_T$  will not be  $c$ -tame. However, as a corollary to Lemma 3.20, we have,

$$\begin{aligned}
 \text{prob}(F_T \text{ isn't } c\text{-tame}) &\leq 2^{-m^c} \\
 &\leq 2^{n^{c'}} \\
 &\leq \frac{1}{n^{c''}},
 \end{aligned}$$

for all constant  $c''$ . Now suppose that an infinitely powerful judge received  $p(n)$  samples of the simulator's output, and  $p(n)$  samples of ZERO-KNOWLEDGE-COMMIT's output, for some polynomial  $p$ . Then with probability at least

$$(1 - 2p(n)/n^{c''}),$$

for any constant  $c''$ , all the outputs,  $O$ , received by the judge, would be output with identical probabilities from both sources. In this case, the judge could not correctly decide which distribution was which with probability greater than  $\frac{1}{2}$ . Thus, it is impossible for the judge to correctly distinguish the two distributions with probability greater than

$$\frac{1}{2} + 2p(n)/n^{c''},$$

for any constant  $c''$ . The lemma follows. ■

### How much information does the commital protocol convey about $X$ ?

The proof of the above lemma actually implies something much stronger. Most of the subtlety involved was in handling the cases where the simulator aborted. In hindsight, these cases were not important, since they occurred with negligible probability. In the case where the simulation actually outputs a triple  $(X \oplus F_X, T \oplus F_T, A^R(T))$ , we have the following facts, which follow almost immediately from the construction of our simulator.

1. There exists a specific set,  $I = \{i_1, \dots, i_q\}$ , such that the simulator asked for the values of  $x_i$ , for  $i \in I$ , and only those values. This set,  $I$ , is a deterministic function of the masks  $F_X, F_T$ .
2. The value of  $(X \oplus F_X, T \oplus F_T, A^R(T))$  output by the simulator induces a conditional distribution on  $X[I] = \{x_i : i \in I\}$ . The conditional distribution the simulator's output induces on  $X$  is equal to the conditional distribution induced by the conditional distribution on  $X[I]$ . That is, no other information about  $X$  is revealed other than that which is revealed by  $X[I]$ .
3. Suppose that the actual commital protocol's output, after going through the oblivious transfer channel, was  $(X \oplus F_X, T \oplus F_T, A^R(T))$ .

Then the conditional distribution induced on  $X[I]$  and  $X$  is the same as that induced when the simulator outputs

$$(X \oplus F_X, T \oplus F_T, A^R(T)).$$

Fact 1 is clear from the construction of the simulator, since the queries are purely functions of  $F_X, F_T$ . Fact 2 is *not* simply due to the fact that the simulator only queried the values of  $X[I]$ . Parts of the tableau simulated would naively seem to depend on parts of  $X$  outside of  $X[I]$ . However, in the proof of the correctness of the MASKED TABLEAU simulator, it is shown that the dependencies of these components, which appear above  $m - 2$  row gaps, are washed out by the randomizing sequence. Fact 3 is implicit in the proof of the indistinguishability between the commital protocol and the simulation.

In other words, suppose that we took the output of protocol ZERO-KNOWLEDGE-COMMIT, which is of the form,

$$(X \oplus F_X, T \oplus F_T, A^R(T)),$$

and computed  $I$  from  $F_X, F_T$ .<sup>2</sup> At this point, we could compute  $X[I]$  from the values of  $X$ , as computed by the protocol, and  $I$ . This is essentially what the simulator does after computing  $F_X, F_T$ , except that instead of computing all of  $X$  beforehand, it just makes the appropriate queries based on  $I$ . Furthermore, the distributions of  $I$ , and hence  $X[I]$  is the same for both the protocol and the simulator. It is this delay in evaluating most of the values of  $X$  that allows us to carry off the simulation.

### Simulating queries of $X$ .

Simulator PARTIAL-ZERO-KNOWLEDGE works fine, provided that some benevolent entity is willing to construct  $X$  according to the correct distribution, and then answer queries about  $X$ . In this section, we show how to simulate such an entity. the difficulty to be addressed is that it is impossible for the simulator to realize the distribution on  $X$  without knowing the values of the committed bits. However, we precisely want to show that a simulation

---

<sup>2</sup>Again, note that  $F_X, F_T$  are implicitly defined by the action of the oblivious transfer channel.

is possible *without* knowing the values of these bits. Thus, we must somehow simulate queries to  $X$  without actually producing  $X$ .

Fortunately, this is actually very easy to accomplish. Our algorithm is very simple: When asked for a bit  $x_i$  of  $X$  that one hasn't been asked before, choose the value of  $x_i$  at random. We now must show that an oracle that makes a random assignment will be, for the purposes of the simulation, indistinguishable from an oracle which produces a correctly distributed  $X$ . To facilitate the argument, we introduce the following terminology.

**Definition 3.10** Fix  $n$  and  $k$ . We define a *query set*,  $I$ , to be a set  $\{i_1, \dots, i_j\}$ , Where  $i_l \in [1, nk]$ . A query set  $I \in \{1, \dots, nk\}^*$  is *well behaved* if for all  $i \in [1, n]$ , we have

$$I_i \not\subseteq I,$$

where  $I_i = [k(i-1) + 1, \dots, ki]$ .

Intuitively, this just says that a well-behaved query set cannot ask for all the bits corresponding to a bit,  $b^i$ . We now show that with high probability, our simulation will only produce a query set which is well behaved.

**Lemma 3.23** Let  $n \leq k^{c_1}$ , for some constant  $c_1$ . Let  $P(x_1, \dots, x_n)$  be an  $NC^1$  predicate, and let  $P_x$  be as defined in Section 3.4. Let  $B_P$  be a canonical W5PBP branching program for  $P_x$ , of length  $m$ , where  $nk \leq m \leq k^{c_2}$ , for some constant  $c_2$ . Then, for any constant,  $c$ , the probability that simulator PARTIAL-ZERO-KNOWLEDGE( $n, k, P$ ) aborts or makes a set of queries that is not well-behaved, is less than  $1/k^c$ , as  $k$  grows sufficiently large.

**Proof:** The simulation only aborts if the mask,  $F_T$ , is not  $c_3$ -tame, for some constant,  $c_3$ , such that  $m^{c_3} \leq \sqrt{k}$ . By the proof of Lemma 3.22, this occurs with negligible probability. We now bound from above the probability that the query set produced by PARTIAL-ZERO-KNOWLEDGE is not well-behaved. The queries produced come in two flavors. First, in Step 3 of the simulation, each element,  $x_i$ , of  $X$  is queried with probability  $1/2$ . Second, in Step 4 of the simulation, at most one query is made to handle each query made by MASKED-TABLEAU. The set of queries made in Step 3 has a very simple distribution, which we will show is, with high probability, "very well-behaved." By very well behaved, we mean that it would take at least  $k/3$  additional queries to make the query set stop being well-behaved. We then observe that the number of queries made in Step 4 is much smaller than this bound.

Denote by  $Q(i)$  the number of queries  $x_j$ , for  $j \in [k(i-1) + 1, ki]$ , made in Step 3 of the simulation. By the binomial theorem, we have the following exact formula for the distribution of  $Q(i)$ .

$$\text{prob}(Q(i) = j) = \binom{k}{j} 2^{-k}.$$

Now, using standard Chernoff bounds, we can bound the probability that  $Q(i) > \frac{2}{3}k$ , for any  $i$ , by

$$\text{prob}(Q(i) > \frac{2}{3}k) \leq e^{-k/18}.$$

Thus, we have

$$\begin{aligned} \text{prob}((\exists i \in [1, n]) Q(i) > \frac{2}{3}k) &\leq n e^{-k/18} \\ &\leq k^{c_1} e^{-k/18}. \end{aligned}$$

Thus, with probability exponentially close to 1, the queries of Step 3 will leave  $k/3$  of the bits representing each committed bit unqueried. But Step 4 only makes a total of  $m^{c_3} \leq \sqrt{k}$  queries. However, for  $k > 9$ ,  $\sqrt{k} < k/3$ , so the query set will still be well-behaved. The lemma follows. ■

### The complete commital simulator.

We can now specify the complete simulator for ZERO-KNOWLEDGE-COMMIT. This simulator will give two outputs. The first output will be the simulation of the commital protocol. The second output will be an ordered pair,  $(I, X[I])$ , where  $I$  is a query set, and  $X[I]$  is the set of constraints placed on  $X$ . This pair will be used to simulate the decommital phase.

The following lemma shows that the simulator given in Figure 3.13 does indeed correctly simulate the commital of  $n$  bits.

**Lemma 3.24** Let  $b^1, \dots, b^n$  be a set of bits, and let  $P$  be some  $NC^1$  predicate. Then, for any  $c$ , and for  $k$  sufficiently large, the distributions produced by protocol ZERO-KNOWLEDGE-COMMIT( $b^1, \dots, b^n, k, P$ ), and by the first component of simulator ZERO-KNOWLEDGE-COMMIT( $n, k, P$ ) cannot be distinguished with probability greater than  $\frac{1}{2} + 1/k^c$ .

**Simulator Zero-Knowledge-Commit( $n, k, P$ )**

- 1: Set  $I = \{\}$ .
- 2: Run PARTIAL-ZERO-KNOWLEDGE( $n, k, P$ ). Handle each query for the value of  $x_i$  as follows:  
  
    **if**  $i \in I$  **then** answer  $X[I]$ .  
    **else** Pick  $b \in \{0, 1\}$  uniformly.  
        Set  $I = I \cup \{i\}$   
        **if**  $I$  is not well behaved, **then** abort.  
        **else** Define  $x_i = b$ .  
            answer  $b$ .
- 3: Simulator **partial-zero-knowledge** will output something of the form  
     $(X \oplus F_X, T \oplus F_T, A^R(T))$ . Output  
  
         $((X \oplus F_X, T \oplus F_T, A^R(T)), (I, X[I]))$ .

Figure 3.13: Simulator for zero-knowledge bit-committal scheme.

By  $k$  being sufficiently large, we require that it be at least  $n^c$ , for some constant  $c > 0$ .

**Proof:** Simulator `ZERO-KNOWLEDGE-COMMIT` is just `PARTIAL-ZERO-KNOWLEDGE` run with a simulation of the  $X$  oracle. By Lemma 3.22, `PARTIAL-ZERO-KNOWLEDGE`, run with a correct oracle for  $X$ , cannot be distinguished from protocol `ZERO-KNOWLEDGE-COMMIT`, with probability

$$\frac{1}{2} + 1/k^{c'},$$

for any  $c'$ .

We now argue that the simulation of the  $X$  oracle will be exact, provided that the query set  $I$  it must answer is well behaved. This is just another restatement of standard parity trick. For a set of committed bits,  $b^1, \dots, b^n$ , the correct distribution of  $X = x_1, \dots, x_{nk}$  is uniform, subject to,

$$(\forall i) \ b^i = \text{parity}(X[I_i]),$$

where  $I_i = [k(i-1) + 1, ki]$ . By definition of a well behaved query set, we have that  $I_i \cap I$  is a proper subset of  $I$ . Hence,  $X[I_i \cap I]$  will be uniformly distributed, which is precisely the distribution of answers realized by the simulated oracle.

Finally, by Lemma 3.23, `PARTIAL-ZERO-KNOWLEDGE` makes queries which are not well-behaved with only exponentially small probability. Therefore, simulator `ZERO-KNOWLEDGE-COMMIT` cannot be distinguished from `PARTIAL-ZERO-KNOWLEDGE`, run with a correct  $X$  oracle, with probability

$$\frac{1}{2} + 1/k^{c''},$$

for any  $c''$ . The lemma follows. ■

### Simulating the decommital protocol.

After having gone through so much work to simulate the commital protocol, the simplicity of simulating the decommital protocol comes as a very welcome relief. The only subtlety is that we use the values of  $I, X[I]$  output by the commital simulator.

To decommit bit  $b^i$ , the decommital protocol merely sends over the values of  $x_j$ , for  $j \in [k(i-1) + 1, ki]$ . The value of  $X = x_1, \dots, x_{nk}$  is completely

**Simulator Zero-Knowledge-Decommit** $(n, k, I, X[I], i, b^i)$

1: Let  $I_i = I \cap [k(i-1) + 1, ki]$ , and write  $X[I_i] = x_{k(i-1)+1}, \dots, x_{ki}$ .  
For  $j \in I \cap I_i$ , let  $x_j$  be as given in  $X[I]$ .

2: For for  $j \in I_i - I$ , choose  $x_j$  uniformly, subject to the constraint,

$$b^i = \text{parity}(X[I_i]).$$

Output  $X[I_i]$ .

Figure 3.14: Simulator for zero-knowledge bit-decommittal.

determined during the committal process, so the decommittal phase is deterministic, and very simple. However, our simulator for the committal phase does not generate a complete description of  $X$ , but merely describes those portions of  $X$  which are relevant to its output. Therefore, our simulator must generate on the fly portions of  $X$ , with the correct distribution, conditioned on the constraints imposed on  $X$  during the committal simulation.

Our simulation is given in Figure 3.14. In the simulation,  $n, k$  are the parameters of the original committal protocol,  $(I, X[I])$  is the second component output by simulator ZERO-KNOWLEDGE-COMMIT,  $i$  specifies which bit is being committed, and  $b^i$  specifies the value of this bit.

It should be remarked that Step 2 may not be well-defined if

$$I_i = [k(i-1) + 1, ki].$$

However, this would imply that the query set  $I$  is not well-behaved. However, all query sets,  $I$ , output by the ZERO-KNOWLEDGE-COMMIT simulator are guaranteed to be well-behaved. We now argue that the simulated distribution produced for committing and decommitting bits is indistinguishable from the actual distribution produced.

**Lemma 3.25** Let  $b^1, \dots, b^n$  be an arbitrary set of bits, and let  $P$  be an  $NC^1$  predicate. Then, as  $k$  grows sufficiently large, the distribution produced by ZERO-KNOWLEDGE-COMMIT( $b^1, \dots, b^n, k, P$ ), followed by running ZERO-KNOWLEDGE-DECOMMIT( $X, i, k$ ) for all  $i \in [1, n]$ , will be indistinguishable from the simulated distribution.

**Remark:** Clearly, if the simulation for decommitting all of the bits is indistinguishable from the actual decommital, then simulating the decommital of any subset of these bits will also be indistinguishable. We are giving the infinitely powerful judge the maximum amount of information it could receive to help in distinguishing the simulation from the real thing.

**Proof:** Deccommitting all the bits corresponds to revealing  $X$ . The action of the simulators can be summarized by the following steps.

1. (committal simulation) Generate,  $F_X, F_T$ , and implicitly  $I$ , with the correct distributions. Occasionally, the simulation aborts, but with only negligible probability, so we will ignore this possibility.
2. (committal simulation) Generate  $X[I]$  according to the correct distribution. Compute  $(X \oplus F_X, T \oplus F_T, A^R(T))$ , which is correctly distributed given  $X[I]$ . Output

$$((X \oplus F_X, T \oplus F_T, A^R(T)), (I, X[I])).$$

3. (decommital simulation) generate and output  $X$ , agreeing with  $X[I]$ .

The action of the actual protocols can be described as the following.

1. (implicit in the oblivious transfer line) Generate,  $F_X, F_T$ , and implicitly  $I$ , with the correct distributions.
2. (committal phase) Generate  $X$ , implicitly generating  $X[I]$ . Compute and output  $(X \oplus F_X, T \oplus F_T, A^R(T))$ .
3. (decommital phase) Output  $X$ .

The distributions of  $((X \oplus F_X, T \oplus F_T, A^R(T)))$ , computed in Step 2 of both the protocol and the simulator, are indistinguishable, by Lemma 3.25.

Nominally, the distributions of  $(X[I], I)$ , generated by the simulator, and implicitly by the protocol, are also indistinguishable. This fact follows from the simulation result for the simple-decommit protocol. Whether one

- Uniformly generates a set of bits  $X[I_i] = x_{k(i-1)+1}, \dots, x_{ki}$ , subject to a parity constraint, and then takes a proper subset of these bits,  $X[I \cap I_i]$ , or,

- Generates  $X[I \cap I_i]$  uniformly, and then generates the rest of  $X_i$ , uniformly subject to the parity constraint,

the resulting distributions on  $X[I_i], X[I \cap I_i]$  are the same. Note that the subsets are indeed proper, since  $I$  is well-behaved, by the construction of the simulator. Note that we are again using the fact that the protocol “generates” values of  $I$  that are not well behaved with only exponentially small probability. The lemma follows. ■

### 3.6 Proving, and improving, aspects of our protocols.

Before concluding this chapter, we finish with a discussion of the reliability, confidence, independence, and security properties of the protocol.

First, we note that the simulation results settle the question of independence and security. Since, for  $k$ , our security parameter sufficiently large, we can simulate ZERO-KNOWLEDGE-COMMIT without knowing the bits that were committed, the security property of our protocol follows. Likewise, since the simulator for the decommital of bit  $b^i$  only needs to know the value of  $b^i$ , the independence property follows.

Second, we can amplify the reliability and confidence of our bit-commitment protocol by running it many times in parallel. Consider the following protocols for secure commitment and decommitment. For our exposition, we let  $C$  be the constant with the following property: If a possibly malicious Alice ran ZERO-KNOWLEDGE-COMMIT, and tried to commit a set of bits,  $b^1, \dots, b^n$  such that  $P(b^1, \dots, b^n)$  was false, Bob would reject with probability at least  $k^{-C}$ , as  $k$  grows sufficiently large.

#### 3.6.1 Analyzing independence, security.

We wish to argue that these protocols have strengthened reliability and confidence properties, while retaining the independence and security properties. First, it is easy to see that the independence and security properties are met. We can run the simulators for commital and decommital in parallel. Such a simulation clearly works, since the protocols are all noninteractive. We give our simulators in Figures 3.17 and 3.18.

**Protocol Secure-Zero-Knowledge-Commit( $b^1, \dots, b^n, k, P$ )**

- 1: Let  $C$  be defined as above. Alice runs ZERO-KNOWLEDGE-COMMIT( $b^1, \dots, b^n, k, P$ ), for  $l = nk^{c+100}$  times in parallel. She remembers  $\mathcal{X} = X_1, \dots, X_l$ , the values of  $X$  returned by each run of the commital protocol.

**Checking Step:** For each run of ZERO-KNOWLEDGE-COMMIT, Bob makes all the checks he would normally make, and rejects if he would have rejected in one of these instances.

Figure 3.15: An improved zero-knowledge bit-commital scheme.

**Protocol Secure-Zero-Knowledge-Decommit( $\mathcal{X}, i, n, k$ )**

- 1: Let  $\mathcal{X} = X_1, \dots, X_l$ . For  $j \in [1, l]$ , Alice runs ZERO-KNOWLEDGE-DECOMMIT on input  $(X_j, i, n, k)$ . Bob will either reject one of the decommitals outright, in which case he rejects, or he will recover bits  $b_1^i, \dots, b_l^i$ . If any of these bits are different, then he rejects, otherwise he reconstructs  $b^i = b_1^i$ .

Figure 3.16: Improved zero-knowledge bit-decommital scheme.

**Simulator Secure-Zero-Knowledge-Commit( $n, k, P$ )**

- 1: With  $l$  defined as in the protocol, run simulator ZERO-KNOWLEDGE-COMMIT( $n, k, P$ ),  $l$  times. For convenience, denote  $\mathcal{I} = I_1, \dots, I_l$ , the set of values of  $I$  output by the simulators. Likewise, write  $\mathcal{X}[\mathcal{I}] = X_1[I_1], \dots, X_l[I_l]$ .

Figure 3.17: Simulator for improved zero-knowledge bit-commital scheme.

**Simulator Secure-Zero-Knowledge-Decommit**  $(n, k, \mathcal{I}, \mathcal{X}[\mathcal{I}], i, b^i)$

With  $l$  defined as above, write  $\mathcal{I} = I_1, \dots, I_l$ , and write  $\mathcal{X}[\mathcal{I}] = X_1[I_1], \dots, X_l[I_l]$ . For  $j \in [1, l]$ , run simulator ZERO-KNOWLEDGE-DECOMMIT( $n, k, I_j, X_j[I_j], i, b^i$ )

Figure 3.18: Simulator for improved zero-knowledge bit-decommittal scheme.

### 3.6.2 Analyzing reliability, confidence.

In discussing reliability and confidence, we consider the possibility of Alice being malicious. She may, for example, wish to commit a set of bits which do not satisfy predicate  $P$ . Alternatively, she may wish to cause Bob to recover bits other than the ones she committed. In any case, she has to work under the following constraints.

1. Alice must implicitly commit a matrix  $B$  of bits,

$$\begin{array}{cccc} b_1^1 & b_1^2 & \dots & b_1^n \\ b_2^1 & b_2^2 & \dots & b_2^n \\ \dots & \vdots & \vdots & \vdots \\ b_l^1 & b_l^2 & \dots & b_l^n \end{array}$$

Decommitting a bit  $b^i$  in the larger protocol corresponds to decommitting column  $i$  of  $B$ . Note, of course, that Alice is not constrained to make  $b_c^i = b_d^i$ , for all  $c, d$ , as would be the case if Alice was good. Furthermore, she can attempt to have Bob reconstruct a bit,  $b_j^i$ , incorrectly, but will be detected with probability at least  $1/2$ .

2. Denote  $B_i = b_i^1, \dots, b_i^n$ . Alice must give a zero-knowledge proof of  $P(B_i)$  for each row  $i \in [1, l]$ . Naturally, Alice may try to give incorrect proofs, but she will be detected with probability  $k^{-C}$ , independently for each incorrect proof she gives.

To facilitate the discussion, we make the following definitions.

**Definition 3.11** Fix predicate  $P$ , and let  $l, c$  be as defined above. Let  $B$  be an  $l$  by  $n$  matrix, as described above. We say that a row  $i$  is *valid* if  $P(B_i)$  is valid. We say that  $B$  is valid if all but at most  $k^{c+1}$  of the  $B_i$ 's are valid. We say that a bit  $i$  is *defined*, with value  $b^i$ , if  $b_j^i \neq b^i$  for at most  $k$  values of  $j$ . Since  $l > 2k$ , the value of a defined bit is unique. If bit  $i$  is not defined, then it is *undefined*.

We now show that Alice can convince Bob to reconstruct bit  $i$  as  $b$  only if bit  $i$  is defined, with value  $b$ .

**Lemma 3.26** [reliability] Consider protocol `SECURE-ZERO-KNOWLEDGE-COMMIT`, run with particular values of  $n, k$ , and  $P$ . Let  $C, l$  be defined as above. Let  $B$  be the matrix of bits implicitly committed by a possibly malicious Alice. Suppose that bit  $i$  is not defined with value  $b^i$ . Then, the probability that, while running `SECURE-ZERO-KNOWLEDGE-DECOMMIT` on bit  $i$ , a possibly malicious Alice can get Bob to reconstruct bit  $b^i$ , is at most  $2^{-k}$ , for  $k$  sufficiently large.

**Proof:** If bit  $i$  is not defined with value  $b^i$ , then there must be at least  $k$  values of  $j$  such that  $b_j^i \neq b^i$ . In order for Alice to get Bob to reconstruct  $b^i$ , she must therefore get him to reconstruct  $k$  bits which were other than what she implicitly committed. However, due to the reliability property of `ZERO-KNOWLEDGE-COMMIT`, Bob has at least a one half chance of rejecting on each of these  $k$  decommitals, independent of whether he accepts or rejects any of the others. The lemma follows. ■

Thus, when a possibly malicious Alice goes through the motions of committing  $n$  bits, we may divide these bits up into two categories. Some bits may be undefined, in which case Alice can hardly ever get Bob to reconstruct any value at all for these bits. The other bits are defined, with set values. Alice can hardly ever get Bob to reconstruct the “wrong” value for these bits. Thus, the protocol is very reliable.

Stating our confidence result is complicated by the fact that some bits may be undefined. We would like to say that if Alice commits a set of bits  $b^1, \dots, b^n$ , such that  $P(b^1, \dots, b^n)$  fails, then Bob will reject with extremely high probability. However, the way the protocol stands, Alice may, for each row  $i \in [1, l]$ , make sure that  $P(B_i = b_i^1, \dots, b_i^n)$  holds, but pick wildly different values for the  $B_i$ 's. In such a case, Bob will *not* reject, even though

some of the “committed” bits may not even be defined. What we in fact need to say is given, and proven, in the following lemma.

**Lemma 3.27** [confidence] Consider protocol SECURE-ZERO-KNOWLEDGE-COMMIT, run with particular values of  $n, k$ , and  $P$ . Let  $C, l$  be defined as above. Let  $B$  be the matrix of bits implicitly committed by a possibly malicious Alice. Then one of the following two cases must hold:

1. Bob rejects with probability at least  $1 - 2^{-k}$  at the checking stage of SECURE-ZERO-KNOWLEDGE-COMMIT.
2. Given  $B$ , one can easily compute a set of bits,  $b^1, \dots, b^n$  such that
  - (a)  $P(b^1, \dots, b^n)$  is true.
  - (b) If bit  $i$  of  $B$  is defined, it has value  $b^i$ .

To prove this result, we consider the case where  $B$  is valid, and the case where  $B$  is not. First, we argue that if  $B$  is not valid, then Bob will reject with high probability.

If  $B$  is not valid, then there must be at least  $k^{c+1}$  rows  $i$ , such that row  $B_i$  is invalid. By definition, Bob will reject each such row with probability  $k^{-c}$ , regardless of whether or not it accepted or rejected any other rows. Hence,

$$\begin{aligned} \text{prob}(\text{Bob doesn't reject}) &\leq (1 - k^{-c})^{k^{c+1}} \\ &\leq \left( (1 - k^{-c})^{k^c} \right)^k. \end{aligned}$$

Now,  $(1 - x^{-1})^x$  is less than  $1/2$ , for  $x$  sufficiently large. Thus, as  $k$  grows sufficiently large, we have

$$\text{prob}(\text{Bob doesn't reject}) \leq \left( \frac{1}{2} \right)^k.$$

We now show that if  $B$  is valid, then one can use it to generate a set of “good” values for all the committed bits, meeting the criteria given in the statement of the lemma.

Assuming that  $B$  is valid, there exist  $l - k^{c+1}$  rows  $j$ , such that  $P(B_j)$  is valid. We will denote this set of rows by  $G$  (for “good”). Now, suppose

for some  $i$  that bit  $i$  is well defined, with value  $b^i$ . Then at most  $k$  rows of  $G$  disagree with  $b^i$  in their  $i$ th position. Therefore, at most  $nk$  rows of  $G$  disagree with any of the defined bits. Since  $l - k^{c+1} - nk > 0$ , there must be at least one row  $j$  such that

$$P(b_j^1, \dots, b_j^n) \text{ is true, and}$$

$$b_j^i = b^i \text{ (for bit } i \text{ defined, with value } b^i).$$

The lemma follows. ■

## Chapter 4

# Circuit Evaluation Using Oblivious Transfer: The $NC^1$ circuit case.

### 4.1 Introduction.

In the previous chapter, we showed how, using an oblivious transfer channel, one can implement very strong commital protocols. Alice can commit a set of bits to Bob, along with a zero-knowledge proof of an arbitrary NP assertions about these bits. Then, Alice can reveal any subsets of these bits, at her discretion, without revealing anything about the other bits. As a byproduct of this commitment technology, Alice can noninteractively prove any NP assertion to Bob.

The problem with all this is that Bob plays the role of the spineless wimp. While Alice does all of the committing, decommitting, and proving, Bob sits passively, making monotonous, routine checks of Alice's integrity. Nobody worries about Bob's integrity; who cares whether a jellyfish is good or evil?

In this chapter, we consider a more equal scenario, which we call *oblivious circuit evaluation*. In oblivious circuit evaluation, Alice and Bob actively work together to compute a function whose arguments are distributed between them. Both Alice and Bob have secret information which they wish to hide from each other. Thus, the scenario is much more symmetric than it was before. As we shall see, some asymmetry will remain in the definitions,

but this will be more due to technical considerations than a reflection of the spirit of the problem.

### **What is Oblivious Circuit Evaluation?**

Let us return to the Alice and Bob story of the previous chapter. Recall that Bob was trying to factor some 500 digit number  $n$ . Alice happened to possess  $d$ , a 100 digit factor of  $n$ , which she hoped to sell to Bob on a bit by bit basis. Using oblivious transfer, she was able to put each of the 100 digits into a magic envelope, and convince Bob that they did indeed represent a factor of  $n$ .

Bob, who had only 25 dollars, was willing to buy 25 of the digits of  $d$ . However, his choice of which digits to buy was somewhat embarrassing. His choice wasn't something logical, such as digits 1 through 25, or 76 through 100. Rather, it consisted of the winning megabucks numbers for the past four days, and the number 17, which was Bob's lucky number. Bob feared persistent, unmerciful teasing at Alice's hands. "Ok, Alice," he said as he handed over the money, "Just give me the envelopes, and I'll go home and pick my 25 digits out."

Alice, who had been born considerably before yesterday, disagreed with this proposal. "How can I be sure you'll only take your fair share. Besides, these envelopes I made can't be opened without my active participation. I simply have to know which digits you want."

"Well, maybe some other time," Bob said sadly. But both knew that "some other time" would never come to pass...

This story would have ended without explaining oblivious circuit evaluation, but for a remarkable turn of events. A few days later, Alice received a Microvax II for her birthday. "Ureka!" she exclaimed. "Our problems are solved."

"How so?" Bob asked.

"Simple. I'll just make a program that will allow you to type in which 25 digits you want to see, give you the answer, and then forget what you asked." Alice was very pleased with herself.

"Well... I might go for this approach." came Bob's hesitant reply. "But I want to see a listing of the program. Just to make sure it gives me the right answer, and really forgets what I asked it. And by the way, are you sure the operating system is secure?"

"Of course, silly," Alice chortled, "It's Unix."

So, Alice and Bob sat down and wrote themselves a program. The program first took a 100 digit number from Alice, and checked that it did indeed divide  $n$ . Then, after clearing the screen, it went through the desired transaction with Bob. There was no way Alice could learn anything about what Bob's input was, and there was no way Bob could learn any more information than what he had paid for.

Life from then on was seemingly idyllic. Alice and Bob soon learned to write even more sophisticated programs. One day, for instance, Alice offered to, for only \$5.95, let Bob know how many how many times a particular digit, of Bob's own choosing, appeared in her divisor. On another day, Alice charged Bob \$10 to learn what value the number had when taken modulo any 8 digit number of Bob's choosing.

But then, after it seemed that all their security problems had been solved, Bob spied a new paperback on Alice's bookshelf. It was titled:

### **How to break Unix, in 10 easy lessons.**

"Alice, I am at a loss for words." he said, and then proved himself wrong by continuing, "How could you have done this? You have destroyed the very basis of all our dealings with each other."

"Bob, I saw the videocassette." Alice replied.

"What videocassette? What are you talking about?" blustered Bob, as beads of sweat started to trickle down his forehead.

"The one so surreptitiously lodged behind *Debbie does Dallas*." Alice answered. "The one titled, *Compromise Unix with Jane Fonda*."

"I didn't know what it was about. It sounded like some kind of kinky, soft-porn flick. You know, like Barbarella. Honest."

Alice was not impressed. Business took a very slow turn from then on. they could still do whatever could be done with the magic envelopes. But the sheer range of deals they could make with the microvax was gone, seemingly forever.

In this chapter, we show how Alice and Bob can, with the use of an ideal oblivious transfer channel, implement oblivious circuit evaluation for  $NC^1$  circuits. In this protocol, Alice and Bob agree on some function,  $F$ , which can be expressed as an  $NC^1$  circuit. Function  $F$  takes two arguments,  $i$  and  $j$ , each of which may consist of some polynomial number of bits. Inputs  $i$  and  $j$  are known to Alice and Bob, respectively. At the end of the protocol, the following three conditions must hold.

1. Bob learns the value of  $F(i, j)$ .
2. Bob only learns  $F(i, j)$ . that is, Bob doesn't learn anything else about the value of  $i$  than can be inferred by knowing the values of  $j$  and  $F(i, j)$ .
3. Alice learns nothing.

### 4.1.1 Outline of the Chapter.

In Section 4.2, we discuss the oblivious string transfer protocols of Brassard-Crépeau-Robert[BCR], and Crépeau[C], and show how to combine it with our zero-knowledge proof commital methodology. In Section 4.3, we show how Alice and Bob can securely evaluate  $NC^1$  circuits.

## 4.2 A Protocol for 1-2-Oblivious String Transfer.

In this section, we discuss a very useful protocol due to the combined work of Brassard-Crépeau-Robert[BCR], and Crépeau[C]. The reason for devoting a section to their work, instead of a simple citation, is three-fold. First, since we use their work as a subprotocol, a description of our protocols would not be complete without it. Second, while using subroutines is very easy and convenient, using subprotocols is a much trickier affair. In order to verify that ones protocol meets all the desired security constraints, one must explicitly construct a simulator for it. However, this task is very hard to accomplish without also constructing simulators for all ones subprotocols. This difficulty is probably more due to the immaturity of the field than to any fundamental theoretical obstacles. Nevertheless, we must live in the present. Third, as will be made clearer, we can write down a slightly simpler, single staged reduction from the problem they solve to ordinary oblivious transfer. This simplification is not due to any conceptual contribution on our part. Historical hindsight allows us to, knowing the final outcome of a series of steps, fabricate a somewhat larger jump that leads directly to the end result.

### 4.2.1 Statement of the Problem.

The problem of *1-2-oblivious string transfer* is conceptually very simple. Suppose that Alice has two strings,  $S_0$  and  $S_1$ , each of length  $n$ .<sup>1</sup> Alice is willing to reveal one, but only one of these strings to Bob. Bob has some bit  $x$ , signifying which string he actually wishes to see. At the end of the protocol, we desire the following security conditions.

1. Bob learns  $S_x$ , except with probability  $< 1/k^c$ , for any  $c$ , where  $k$  is the security parameter.
2. Bob gains no information about  $S_{1-x}$ , except with probability  $< 1/k^c$ , for any  $c$ , where  $k$  is the security parameter.
3. Alice learns nothing.

Brassard-Crépeau-Robert[BCR] first reduced this problem to the case where each of Alice's strings consist of only 1 bit. This case is known as *1-2-oblivious transfer*. Crépeau[C] then reduced *1-2-oblivious transfer* to ordinary oblivious transfer. For simplicity's sake, we present a single-staged reduction to oblivious transfer.

### 4.2.2 The Protocol.

The protocol in Figure 4.1 implements 1-2-oblivious string transfer using ordinary oblivious transfer. As usual,  $k$  denotes our security parameter. In order to make all our security requirements hold, we require that  $k \geq n^c$ , for some constant  $c$ . For convenience, we shorten the name of our oblivious string transfer protocol to OST.

#### Does Bob learn $S^x$ ?

We wish to say that that Bob can recover  $S^x$ , without any error, with extremely high probability.

---

<sup>1</sup>We may alternatively require that  $|S_0|, |S_1| \leq n$ , a slightly more general scenario. Using a simple encoding scheme, this may be easily reduced to the problem with fixed length strings.

**Protocol OST( $n, k$ )**

**0:** We denote Alice's secrets by  $S_0, S_1$ , and Bob's selector bit by  $x$ .

**1:** Alice makes the following computations.

**1.1:** We write  $S_i = s_1^i s_2^i \cdots s_n^i$ , for  $i \in \{0, 1\}$ . Alice then chooses  $2nk$  bits,  $s_{j,l}^i$ , for  $i \in \{0, 1\}, j \in [1, n]$ , and  $l \in [1, k]$ . These bits are chosen uniformly, subject to

$$s_j^i = \bigoplus_{l=1}^k s_{j,l}^i.$$

**1.2:** Alice uniformly chooses two bijections,

$$\Pi_0, \Pi_1 : [1, n] \times [1, k] \longrightarrow [1, nk].$$

**1.3:** Alice uniformly chooses  $3nk$  bits  $B = b_1, \dots, b_{3nk}$ .

**2:** Alice transfers  $B$  to Bob, using the oblivious transfer channel.

**3:** Bob computes  $A_0, A_1$ , two disjoint subsets of  $[1, 3nk]$ . We write  $A_i = a_1^i, \dots, a_{nk}^i$ . We require that  $|A_0| = |A_1| = nk$ , and that Bob received bit  $b_{a_j^x}$ , for  $j \in [1, nk]$ . If no such sets exist, then Bob aborts. Otherwise, Bob sends  $A_0, A_1$  to Alice.

**4:** For notational convenience, we adopt the notation

$$\langle i, j, l \rangle = a_{\Pi_i(j,l)}^i.$$

For all  $i \in \{0, 1\}, j \in [1, n]$ , and  $l \in [1, k]$ , Alice sends Bob,

$$s_{j,l}^i \oplus b_{\langle i,j,l \rangle}.$$

Alice also sends Bob  $\Pi_0$ , and  $\Pi_1$ .

**5:** Bob uses the bits given in Step 4 to reconstruct  $s_{j,l}^x$ , for  $j \in [1, n]$ , and  $l \in [1, k]$ . He then reconstructs  $S_x = s_1^x s_2^x \cdots s_n^x$ , for  $j \in [1, n]$ , by computing the appropriate parity function(given in Step 1.1).

Figure 4.1: Protocol for oblivious string transfer([BCR],[C]).

**Lemma 4.1** At the conclusion of  $\text{OST}(S_0, S_1, x, k)$ , Bob completely recovers  $S^x$ , with probability at least  $1 - 1/k^c$ , for all constant  $c$ .

**Proof:** Bob will always be able to reconstruct  $S^x$ , provided that he can successfully create the sets  $A_0$  and  $A_1$  in Step 3 of the protocol. This can be done if and only if Bob actually received at least  $nk$  of the  $3nk$  bits. By a simple application of Chernoff bounds, we have

$$\begin{aligned} \text{prob}(\text{Bob gets less than } nk \text{ bits}) &\leq e^{-(nk)^2/(2 \cdot 3nk)} \\ &\leq e^{-\frac{nk}{6}} \end{aligned}$$

The lemma follows. ■

### Does Alice learn anything?

It is not hard to see that if Bob follows the protocol, then Alice will gain no information about  $x$ . All Alice receives from Bob is an ordered pair of sets. One of these sets refers entirely to bits which Bob received; one does not. However, since Alice has no idea which bits actually got through, this tells her nothing. Given Alice's knowledge, either set is equally likely to refer to bits which got through to Bob. Note that there is no chance, not even an exponentially small one, of Alice getting any information. Such a distinction between exponentially small and 0 is not in the spirit of this thesis, but may be of slight theoretical interest.

One might also consider whether Alice can gain any advantage by cheating. However, Bob's message to her depends only on the channel, and not on anything she does. Furthermore, any messages she sends are either clearly incorrect, say having the wrong number of bits, or correspond to some legitimate choice of  $S_0$  and  $S_1$ . Since Alice, playing legitimately, can pick whatever  $S_0, S_1$  she likes, this does not confer any extra power on her. All she can do by cheating is to perhaps give Bob some extra information about the string he did not select. Thus, there is no point in Alice breaking the protocol if she is willing to play in the first place.

### How much does Bob learn?

The crucial task before us is to formally show that Bob does not learn anything about string  $S_{1-x}$ . This is done by the standard, painful exercise of

explicitly constructing a simulator. An issue which must be addressed, however, is that of exactly what powers the simulator should be allowed to have. With our commital protocols, we constructed a simulator which ran in probabilistic polynomial time. This made sense, since we wanted to show that the zero-knowledge proofs transmitted absolutely no information. However, in this case, some information is definitely transmitted, namely the string  $S_x$ . Since we are making no assumptions about Alice or the strings  $S_i$ , we must seemingly allow our simulator to have arbitrary computational power.

We would like to say that running our oblivious string transfer protocol is somehow equivalent to using some ideal “black box” protocol which meets the desired security specifications. To show this, we consider the following scenario. Imagine that Alice is willing to reveal exactly one of her two strings. Since we are bounding Bob’s knowledge, it is kosher to allow the simulator to reveal its choice to Alice. The simulator is a probabilistic polynomial-time machine which can query Alice for exactly one of these strings. The simulator also talks to Bob, who is running (perhaps maliciously) his side of the OST protocol. The job of the simulator is to simulate Alice running her side of the protocol, through a simulated oblivious transfer channel. The goal of such a simulation is to

Now, if such a simulator exists, then Bob can not learn anything more from running the protocol, even maliciously, than he could obtain through running some ideal oblivious string transfer protocol with Alice. In particular, this would imply that he learns nothing about  $S_{1-x}$ , where bit  $x$  is implied by Bob’s actions.

Another way of looking at this is that the simulator acts as a buffer between a trustworthy Alice and a malicious Bob. Instead of having the malicious Bob speak to Alice directly, we have him speak to the simulator, who then interacts with Alice, *through an ideal protocol*. Bob cannot tell whether he was talking directly to Alice, or to the simulator. This means that any information he could steal from Alice, he could learn from participating in the ideal protocol, and running the simulator himself.

The simulator we present will occasionally abort the simulation, needing more information than can be obtained by running the ideal protocol with Alice. In these cases, we make no guarantees about how much information Bob obtains. However, as will be shown, the simulator aborts with only exponentially small probability. Thus, as far as we are concerned, this possibility may be safely ignored.

### 4.2.3 The Simulator.

Our simulator, given in Figure 4.2 is actually quite straightforward. First, it simulates sending the  $3nk$  random bits to Bob. When Bob sends back his two sets, it determines which string, if any, Bob has any hope of learning. Then it queries Alice for this string. Finally, it computes, in a fairly straightforward manner, its final transmission to Bob.

As usual, we ignore the possibility that Bob may send back something really weird. We implicitly assume that any party aborts the protocol if it receives a transmission that doesn't remotely resemble a legal message. This behavior is very easy to simulate, and in the interest of clarity, we omit further mention of it.

#### Is this a faithful simulation?

Before we show anything about the distributions produced by the simulators, we first show that the simulator rarely aborts.

**Lemma 4.2** Let  $k$  be at least  $n^{c'}$  for some  $c' > 0$ . Let Bob use any “reasonable” strategy (i.e. Bob never says anything that would cause Alice to immediately abort - e.g. giving messages of the wrong length). Then simulator  $\text{OST}(n, k)$  will abort with probability less than  $1/k^c$ , for any  $c$ .

**Proof:** The simulation aborts (nontrivially) under the following two circumstances:

1.  $F$ , the mask it uses to simulate the oblivious transfer channel, has more than  $7nk/4$  ones.
2. there exists some  $j$  such that

$$(\forall l) f_{\langle \bar{x}, j, l \rangle} = 1.$$

The first case can be routinely dealt with using a standard Chernoff bound argument. We can bound the probability that this case occurs by  $e^{-\Omega(nk)}$ . Hence, for the purposes of this lemma, we can assume that this condition is always met.

The second case is also easy, though slightly more complicated to write out. Now, we can assume that the number of ones in  $F$  is no more than

**Simulator OST( $n, k$ )**

- 1: The simulator makes the following computations.
  - 1.1 The simulator uniformly chooses two bijections,  $\Pi_0, \Pi_1$ , from  $[1, n] \times [1, k]$  to  $[1, nk]$ .
  - 1.2 The simulator uniformly chooses  $3nk$  bits,  $B = b_1, \dots, b_{3nk}$ .
  - 1.3 The simulator uniformly chooses a  $3nk$  bit mask,  $F = f_1, \dots, f_{3nk}$ . If the number of ones in  $F$  exceeds  $7nk/4$ , then the simulator aborts.
- 2: The simulator sends  $B \oplus F$  to Bob. Bob sends back two disjoint sets  $A_0, A_1$ , each of size  $n$ . We write  $A_i = a_1^i, \dots, a_{nk}^i$ , where  $a_j^i \in [1, 3nk]$ .
- 3: For  $i \in \{0, 1\}$ , The simulator computes  $e_i = |\{j : f_{a_j^i} = 1\}|$ . It then calculates  $x$ , such that  $e_x \geq e_{1-x}$ . In other words, the simulator picks the  $x$  corresponding to which set  $A_x$  refers to the most bits that were actually received.
- 4: The simulator asks Alice for her value of  $S_x$ . We write  $S_x = s_1^x s_2^x \dots s_n^x$ . The simulator uniformly chooses  $S_{\bar{x}} = s_1^{\bar{x}} s_2^{\bar{x}} \dots s_n^{\bar{x}}$ .
- 5: The simulator chooses  $2nk$  bits,  $s_{j,l}^i$ , for  $i \in \{0, 1\}, j \in [1, n]$ , and  $l \in [1, k]$ . These bits are chosen uniformly, subject to

$$s_j^i = \bigoplus_{l=1}^k s_{j,l}^i.$$

- 6: If there exists some  $j$  such that

$$(\forall l) f_{\langle \bar{x}, j, l \rangle} = 1,$$

then the simulator aborts. Otherwise, the simulator sends Bob the value of

$$s_{j,l}^i \oplus b_{\langle i, j, l \rangle},$$

for all  $i \in \{0, 1\}, j \in [1, n]$ , and  $l \in [1, k]$ . Also, the simulator sends Bob  $\Pi_0$  and  $\Pi_1$ .

Figure 4.2: Simulator for oblivious string transfer protocol.

$7nk/4$ . Then, by the definition of  $x$ , we have that  $e_{1-x} \leq 7nk/8$ . In other words, there are at least  $nk/8$  values of  $m$  such that

$$f_{a_m^x} = 0.$$

We now show that for any  $j \in [1, n]$ , and for  $\Pi_{1-x}$  chosen uniformly, the probability that,

$$f_{\langle \bar{x}, j, l \rangle} = 1,$$

for all  $l \in [1, k]$ , is exponentially small. Since  $\Pi_{1-x}$  is completely random, the above probability is equal to the probability that

$$f_{a_m^{1-x}} = 1,$$

for  $k$  distinct random values of  $m \in [1, nk]$ . That is, the value of  $m$  is chosen  $k$  times from  $[1, nk]$ , without replacement. This probability will only be increased if we allow  $m$  to be chosen with replacement. However, since at least  $1/8$  of the  $m$ 's violate the condition, the chance of none of them violating it is at most  $(7/8)^k$ . The lemma follows. ■

We now show that the above simulator will give a faithful simulation, provided that  $k$  is at least  $n^c$  for some  $c > 0$ .

**Lemma 4.3** Let  $k$  be at least  $n^c$  for some  $c > 0$ . Then the distribution of conversations between the simulator of  $\text{OST}(n, k)$  and Bob will be indistinguishable from conversations between Alice and Bob running protocol  $\text{OST}(n, k)$ . This holds regardless of Bob's strategy (which is allowed to differ from the protocol), and Alice's choice of  $S_0$  and  $S_1$ .

**Proof:** The simulation can differ from the actual protocol in two ways.

1. The simulation aborts under certain situations, when the real protocol would not.
2. Given some strategy of Bob's, and some value of  $S_0, S_1$ , the ensemble of conversations with the simulator may differ from those with Alice.

The first case is handled by Lemma 4.2. The second case is handled almost by inspection. Virtually everything computed by the simulator is computed in the exact same way as by the actual protocol. The only difference is that

a random  $S_{1-x}$  is used instead of the actual one. However, assuming that the simulation doesn't abort, we know that

$$(\forall j)(\exists l)f_{\langle \bar{x}, j, l \rangle} = 0.$$

Thus, for each  $j$ , there is an  $l$  such that

$$b_{\langle \bar{x}, j, l \rangle}$$

was never “received” by Bob. Hence, the value of

$$s_{j,l}^{\bar{x}} \oplus b_{\langle \bar{x}, j, l \rangle}$$

will be uncorrelated with anything visible to Bob, and in particular, it will be uncorrelated with  $s_{j,l}^{\bar{x}}$ . The values of  $s_{j,1}^{\bar{x}}, \dots, s_{j,k}^{\bar{x}}$  computed by the simulator are uniformly distributed, and the values that would be computed by Alice are distributed uniformly subject to a parity constraint. However, since only a proper subset of these bits are seen, the distributions are indistinguishable. Thus, whether we used the correct values of the  $s_j^{\bar{x}}$ 's or random ones, or even all 0's, the distribution will still be the same. The lemma follows. ■

What does this lemma really say? One way of looking at it is that if we were to modify a protocol, replacing the use of OST with Bob just querying Alice for one of the strings, the knowledge given to Bob would be the same(except with negligible probability). This of course assumes that the OST protocol is being used as essentially a black box, with Alice forgetting about what happened during the protocol. In other words, if the protocol were to have Alice's later behavior depend on, for instance, which particular values of the  $s_{j,l}^i$ 's she chose, then conceivably Bob could gain extra information.

Alternatively, another way of looking at this lemma is that Bob might as well obey the protocol. Instead of committing the *faux pas* of actually cheating, Bob could simply think about cheating. He could then with high probability simulate this cheating while legitimately running the protocol with Alice.

### Running OST in parallel.

It is worth noting that this simulation can be run in parallel. With high probability, the simulator can keep on running, and give accurate simulations, regardless of whatever Bob tells it. Thus, even with several Bob's

conspiring, they cannot block the simulation. Often, more conventional simulations ([GMR],[GMW]), have to start over whenever Bob gives an awkward question. Bob will give a nonawkward question with non-negligible probability, so these strategies work for a single Bob. However, when one tries to run these simulations in parallel, the probability that all the Bob's ask non-awkward questions at the same time becomes exponentially small. In some sense, the standard simulations are pathological, since intuitively one should be able to run zero-knowledge protocols in parallel - lots of 0's in parallel should still be 0!

We adopt the convention that whenever we run OST several times in parallel, we run them in lock step: Alice gives all her messages for Step 2 at once, and all her messages for Step 4 at once. Thus, Alice doesn't give Bob any information about any of the strings until Bob has finished sending messages. Likewise, when we run the simulations in parallel, we have the simulator ask Alice for all the strings at once. This convention will turn out to be useful in later protocols.

#### 4.2.4 An application: Oblivious decommittal.

We have described a protocol which allows Bob to select exactly one of 2 strings given by Alice, such that Alice doesn't know which string Bob selected. This is by itself a very useful capability. However, we can do even more by combining the oblivious string transfer protocol of this chapter with the commital/decommittal protocols of the previous chapter. Recall that Alice decommits a bit by simply giving out a particular string she decided on in the commital phase. Therefore, we can have the following scenario. Alice has committed a set of bits,  $b^1, \dots, b^n$ , possibly with a zero-knowledge proof of some assertion about these bits. Let  $I_0, I_1$  be subsets of  $[1, n]$ . Alice is willing to either

- Decommit bits  $b_i$ , for  $i \in I_0$ .
- Decommit bits  $b_i$ , for  $i \in I_1$ .

How would she do this? Well, in either case, she would simply release to Bob a set of strings, one for each bit she wished to decommit. These sets of bit strings may be thought of as one big string, so in either case, Alice is really just sending a big string to Bob. We can call these strings  $S_0, S_1$  respectively.

**Protocol Oblivious-Decommit( $X, k, I_0, I_1$ )**

- 1: Alice computes strings  $S_0, S_1$ , by making the appropriate calls to ZERO-KNOWLEDGE-DECOMMIT.
- 2: Alice and Bob run OST( $n, k$ ), where Alice's secret strings are  $S_0$  and  $S_1$ , and Bob's selector bit is  $x$ . Bob uses  $S_x$  to reconstruct the committed bits whose locations are in  $I_x$ . He does this by taking each substring of  $S_x$  which corresponds to a single bit decommital, and treating it as if he had been sent the string by the usual decommital protocol. If he detects some inconsistency, he does not assign any value to the decommitted bit. However, he does not tell this fact to Alice.

Figure 4.3: Protocol for oblivious decommital.

Now, suppose Bob wished for Alice to decommit the bits referred to in  $I_x$ , for some  $x$ , but did not wish for her to know his value of  $x$ . All they have to do is run the oblivious string transfer protocol, with strings  $S_0, S_1$ . We give the protocol and its simulator in Figures 4.3 and 4.4.

**Remark:** The reason Bob ignores bad strings is to protect his value of  $x$ . Running OST clearly gives Alice no information about  $x$ , but seeing whether Bob received a consistent string could give Alice information. For instance, she could make  $S_0$  complete garbage, and determine  $x$  from whether Bob complained. With Bob sufficiently taciturn, Alice gets no information about  $x$  by the same argument as for protocol OST.

The simulator for this protocol is equally trivial. Our scenario is as follows. The simulator talks to Alice, who is willing to reveal one of two groups of bits. The simulator also talks to a possibly malicious Bob, and attempts to simulate the oblivious-decommital protocol run with him.

It is not hard to see that the oblivious decommital protocol and its simulator have the desired properties, provided that  $k$  is sufficiently large. All the simulator is doing is making calls to Alice and other simulators whose faithfulness has already been established. Furthermore, they may be run and simulated in the same lockstep style as with OST. Also, it is clear that Alice cannot trick Bob into recovering a bit other than what was committed - she has no advantage over the usual method of decommitting.

**Simulator Oblivious-Decommit( $I, X[I], k, I_0, I_1$ )**

- 1: We write  $X[I] = \{x_i\}$ , for  $i \in I$ . The simulator runs the simulator for  $\text{OST}(n, k)$ , where  $n$  is the length of the concatenated decommittal strings. When the simulator asks for string  $S_x$ , the simulator asks Alice to reveal the values of bits  $x_i$ , for  $i \in I_x$ .
- 2: For each location  $i \in I_x$ , Alice runs simulator  $\text{ZERO-KNOWLEDGE-DECOMMIT}(N, k, I, X[I], i, x_i)$ , where  $N$  is the total number of bits committed.
- 3: The simulator concatenates the decommittal strings into a single string,  $S_x$ , which it hands to the OST simulator.

Figure 4.4: Simulator for oblivious decommittal.

#### 4.2.5 Incorporating OST and oblivious-decommittal simulators into larger simulations.

As we will use OST and OBLIVIOUS-DECOMMIT in larger protocols, it is essential that we be able to use their simulators as subroutines in the larger simulations. It is not immediately obvious that one can do this for arbitrary simulators, since the simulators' primary purpose is to bound knowledge transfer. Fortunately, they can indeed be used, in a relatively straightforward manner.

The simulators we've constructed in this chapter all make queries to some friendly Alice, who is in effect running some ideal version of OST {OBLIVIOUS-DECOMMIT}. Of course, the larger simulations may not have a friendly Alice at their beck and call. Or, as will be more often the case, they will have an Alice to whom they can make limited queries, but who will not help them with such details. However, they may be able to simulate Alice's responses to the OST {OBLIVIOUS-DECOMMIT} simulator's queries. If so, then the simulation of the OST {OBLIVIOUS-DECOMMIT} phase will be correct.

Now, often it will be the case that in some protocol, Alice will perform a lot of oblivious string transfers, and the pairs of strings she uses may not be easily simulated. However, the OST simulator will only ask for one of

these strings, and it is often substantially easier to simulate the answers of these queries than to simulate the actual pairs. For instance, consider the following protocol fragment.

**17:** Alice, having finally computed bit  $b$ , which is 1 iff God exists, uniformly chooses some random bit,  $r$ . She then runs  $\text{OST}(b \oplus r, r, k)$ .

It may well be very difficult to simulate the pair  $(b \oplus r, r)$ . However, one can still simulate step 17 by

**1:** Choose bit  $r'$  uniformly. Run simulator  $\text{OST}(1, k)$ .

**2:** When the OST simulation makes query,  $x$ , answer with  $r'$ .

In this example, seeing one of the two bits is equivalent to seeing a random bit, so the simulation goes through. Note also that in the above example, the higher level simulator was able to know which bit the OST simulator requested. This might seem magical, but really isn't. The crux of the OST simulation is that one knows which bits the (simulated) oblivious transfer channel sent to Bob, since one is the one simulating the channel. With this extra information, one can determine which, if any, of the two strings Bob will be able to learn anything about.

### 4.3 Oblivious Evaluation of $NC^1$ Circuits.

Before tackling the general problem of oblivious circuit evaluation, we first consider a restricted version. Instead of working with arbitrary polynomial-sized circuits, we confine ourselves to log-depth polynomial-sized circuits, i.e. those circuits in  $NC^1$ . In this section, we describe a protocol which allows Alice and Bob to securely compute  $F(i, j)$ , where  $F$  is an  $NC^1$  circuit.

To do this, we use Barrington's theorem on the equivalence between W5BPB's and  $NC^1$  circuits. Our general strategy is to first convert our circuit into a set of branching programs, and then randomize our branching program so as to intermediate computations (this will become clearer in the next section). These randomized branching programs can be easily computed using the oblivious string transfer or oblivious decommittal protocols described in the last section.

Our exposition will be first given under the (incorrect) assumption that Alice may be trusted. It will turn out to be very easy to convert our protocols to ones in which Alice may be malicious, but is forced to behave honestly (or risk having Bob abort the protocol).

### 4.3.1 Simplifying our circuits.

For the moment, we will confine our attention to circuits which have only one bit as output, and are thus amenable to Barrington's transformation. Later, we will show how to increase the number of bits of output. Thus, we are really concerned with how to obviously evaluate W5PBP's.

### 4.3.2 Randomizing W5PBP's.

Recall that a W5PBP,  $B$ , with inputs  $x_1, \dots, x_k$ , is of the form,

$$B = ((i_1, \pi_1^0, \pi_1^1), (i_2, \pi_2^0, \pi_2^1), \dots, (i_m, \pi_m^0, \pi_m^1), a),$$

where  $i_j \in [1, k]$ ,  $\pi_j^i \in S_5$ , and  $a \in S_5, a \neq I$ . To evaluate  $B(x_1, \dots, x_k)$ , we simply evaluate

$$\prod_{j=1}^m \pi_j^{x_{i_j}}.$$

If this product is equal to  $a$ , the output is defined to be 1. If the product is equal to the identity, the output is 0. It is required by the definition of W5PBP's that  $a$  and  $I$  are the only possible products.

We now define the notion of a *randomized* W5PBP.

**Definition 4.1** Let  $B$  be a length  $m$  W5PBP, with notation as above, and let  $R = r_1, \dots, r_{m-1}$  be a sequence of elements of  $S_5$ . We define the *randomized W5PBP*,  $B^R$  by

$$B^R = ((i_1, \pi_1'^0, \pi_1'^1), (i_2, \pi_2'^0, \pi_2'^1), \dots, (i_m, \pi_m'^0, \pi_m'^1), a),$$

where

$$\begin{aligned} \pi_1'^i &= \pi_1^i r_1^{-1}, \\ \pi_m'^i &= r_{m-1} \pi_m^i, \text{ and,} \\ \pi_j'^i &= r_{j-1} \pi_m^i r_j^{-1}, \end{aligned}$$

for  $1 < j < m$ .

Essentially, we are doing the same randomization of permutation sequences we did in proving the gap-randomization property for randomizing tableaux. Now, if we take the product

$$\prod_{j=1}^m \pi_j^{x_{i_j}},$$

for any valuation of the  $x$ 's, and expand out the values of the  $\pi_j^i$ 's, we get a telescoping product, in which all the  $r$ 's cancel out. This is essentially the same argument as in the proof of Lemma 3.9, so we will not expound on it with our usual gusto. This implies that  $B^R$  is a legitimate branching program, and furthermore that

$$B^R(x_1, \dots, x_k) = B(x_1, \dots, x_k).$$

#### A useful property of randomized W5PBP's.

As we saw, randomizing a branching program will in no way affects its output. Therefore, one might ask why one would want to have them. The reason is that, in a certain technical sense, sufficiently randomized W5PBP's hide all partial information about their intermediate calculations. We make this statement more precise, as follows.

**Lemma 4.4** (the W5PBP Security Lemma) Let  $B$  be a length  $m$  W5PBP, with notation as above. Let  $x_1, x_k$  take on some arbitrary set of values. Let  $R = r_1, \dots, r_{m-1}$  be a uniformly distributed randomizing sequence. Let

$$A' = \pi_1^{x_{i_1}}, \dots, \pi_m^{x_{i_m}},$$

be the sequence of permutations obtained by evaluating  $B^R$ . Then  $A'$  will be uniformly distributed over length  $m$  sequences of elements in  $S_5$ , subject to

$$\prod_{j=1}^m \pi_j^{x_{i_j}} = \begin{cases} a & \text{if } B(x_1, \dots, x_k) = 1, \\ I & \text{if } B(x_1, \dots, x_k) = 0. \end{cases}$$

**Notation:** We refer to a W5PBP that is randomized with a uniformly distributed randomizing sequence as a *uniformly randomized W5PBP*.

**Proof:** Referring to Lemma 3.11, we see by inspection that

$$A' = \text{scramble}_A(R),$$

where,

$$A = \pi_1^{x_{i_1}}, \dots, \pi_m^{x_{i_m}}.$$

The lemma follows immediately. ■

What does this mean in terms of security? It means that if Bob sees the sequence of permutations used to evaluate some uniformly randomized W5PBP, it tells him the final answer, and nothing more. This idea is the fulcrum of our protocol for evaluating  $NC^1$  circuits.

### 4.3.3 The Protocol for W5PBP Evaluation.

Let  $B$  consist of the triples  $(i_j, \pi_j^0, \pi_j^1)$ , for  $j \in [1, m]$ , along with an accept state,  $a$ . Let us assume that Bob possesses inputs  $x_1, \dots, x_l$ , and Alice possesses inputs  $x_{l+1}, \dots, x_n$ . We wish for Bob to be able to determine the value of  $B(x_1, \dots, x_n)$ , without learning anything more, and without leaking any information to Alice.

The following definitions are useful for the statement of our protocol.

**Definition 4.2** Let  $B$  be a W5PBP defined as above. For  $i \in [1, n]$ , we define the sequence  $var^i(B) = var_1^i(B), var_2^i(B), \dots$  by

$$var_1^i(B) < var_2^i(B) < \dots,$$

and  $j \in var^i(B)$  iff  $i_j = i$ . We define the sequence of pairs,

$$V_i(B) = (V_i^0(B), V_i^1(B)),$$

by,

$$V_i^j(B) = \pi_{var_1^i(B)}^j, \pi_{var_2^i(B)}^j, \dots.$$

We denote by  $B_{vars}$  the sequence  $i_1, \dots, i_m$ .

The sequence  $var^i(B)$  simply consists of the indices of all the triples in  $B$  that refer to variable  $x_i$ . The sequence  $V_i^j(B)$  consists of all the permutations that appear in the evaluation of  $B$  when  $x_i = j$ . Thus, one way of computing  $B(x_1, \dots, x_n)$  is to first compute  $V_i^{x_i}(B)$ , for  $i \in [1, k]$ , and then multiply these permutations in the standard order. The point here is that these sequences contain all the permutations we need.

Our first protocol, given in Figure 4.5, is very naive in that it's security relies on Alice obeying the protocol. We will see that our commital technology makes it trivial to upgrade the protocol to work even when Alice isn't honest.

**Protocol Honest-W5PBP-Eval( $B, l, n, k$ )**

- 1: Alice uniformly chooses a randomizer  $R$ , and computes  $B^R$ . Alice then computes  $V_i(B^R)$  for  $i \in [1, n]$ .
- 2: For  $i \in [1, l]$ , Alice and Bob execute, in parallel,  $\text{OST}(V_i^0(B^R), V_i^1(B^R), x_i, k)$ .
- 3: For  $i \in [l + 1, n]$ , Alice sends Bob  $V^{x_i}(B^R)$ . Bob now uses this information to reconstruct  $B(x_1, \dots, x_n)$ .

Figure 4.5: Protocol for oblivious W5PBP evaluation (honest case).

**Bounding Alice's information.**

It is not hard to see that Alice receives no information about Bob's variables from this protocol. This follows from the fact that OST gives no information to Alice.

**Bounding Bob's information.**

We would like to show that Bob receives no more information than he could if he simply gave Alice his arguments, and had Alice return  $B(x_1, \dots, x_n)$  to him. To show this, we construct a simulator which communicates with Bob, and simulates Alice running her side of the protocol. At some point, the simulator will give Alice its arguments, and receive the value of the branching program. We give our simulator in Figure 4.6.

We would like to build on our previous simulator work, namely the simulator for OST. Recall that the simulator for OST is guaranteed to provide a faithful simulation, but has to talk to a friendly Alice, to whom it makes its requests. The real Alice is only willing to play the ideal W5PBP evaluation game. However, we can have our new simulator run the OST simulations, and simulate this friendly Alice to them (given some help from the real Alice). This is a big win, since we can use the OST simulators as a sort of buffer. Whereas a malicious Bob may perform in extremely bizarre, though ultimately ineffectual ways, the behavior of the OST simulator is standardized: No matter how Bob behaves, the OST simulator merely asks Alice for one of two particular strings.

**Simulator Honest-W5PBP-Eval( $B_{vars}, l, n, k$ )**

- 1: Let  $q_i$  be the length of  $V_i^0(B)$  (note that this equals  $V_i^j(B^R)$ , for any randomizer  $R$ , and  $j \in \{0, 1\}$ ). For  $i \in [1, l]$ , (where Bob contributes  $x_1, \dots, x_l$ ) the simulator executes, in parallel, subsimulator  $OST(q_i, k)$ . Subsimulator  $OST$  will, in the course of its execution, make  $l$  queries to the simulated Alice, which we denote  $x_1, \dots, x_l$ . The simulator sends  $x_1, \dots, x_l$  to Alice, who returns  $B(x_1, \dots, x_n)$ .

- 2: The simulator uniformly chooses  $\pi_1, \dots, \pi_m$ , subject to

$$\prod_{i=1}^m \pi_i = \begin{cases} a & \text{If } B(x_1, \dots, x_n) = 1 \\ I & \text{If } B(x_1, \dots, x_n) = 0 \end{cases}.$$

For  $j \in [1, n]$  the simulator computes

$$V_i^{sim} = \pi_{var_1^i(B)}, \pi_{var_2^i(B)}, \dots.$$

- 3: The simulator gives the  $OST$  subsimulator,

$$V_1^{sim}, \dots, V_l^{sim},$$

as answers to its queries.

- 4: The simulator sends Bob  $V_{l+1}^{sim}, \dots, V_n^{sim}$ .

Figure 4.6: Simulator for oblivious W5PBP evaluation (honest case).

We now show that this simulation is correct when the security parameter is sufficiently large.

**Lemma 4.5** Let W5PBP  $B$  be as above, and let  $k \geq (nm)^c$ , for some  $c > 0$ . Then for any strategy Bob employs, the output of simulator HONEST-W5PBP-EVAL( $B, l, n, k$ ) will be indistinguishable from the output of protocol HONEST-W5PBP-EVAL( $B, l, n, k$ ).

**Proof:** Given the bound on  $k$ ,  $k$  is guaranteed to be at least  $q_i^c$  for all  $i \in [1, l]$ , and some  $c > 0$ . Hence, by Lemma 4.3; the simulations of OST will be indistinguishable from the actual runs, provided that the answers given by the simulated Alice are correctly distributed.

The actual Alice runs OST with the values  $V_i^0(B^R)$  and  $V_i^1(B^R)$ , for  $I \in [1, l]$ . Thus, its answers to the simulator's queries would be of the form

$$V_1^{x_1}(B^R), \dots, V_l^{x_l}(B^R),$$

where the distribution on simulator queries,  $x_1, \dots, x_l$ , have some distribution which depends on Bob's strategy. Also, Alice gives Bob the values of

$$V_{l+1}^{x_{l+1}}(B^R), \dots, V_n^{x_n}(B^R),$$

where  $x_{l+1}, \dots, x_n$  are set by Alice.

What is this view? Writing

$$B^R = ((i_1, \pi_1^0, \pi_1^1), \dots, (i_m, \pi_m^0, \pi_m^1), a),$$

the set of transmissions sent by Alice is equivalent to the set of permutations

$$\pi_j^{x_{i_j}},$$

for  $j \in [1, m]$ . But by the W5PBP Security Lemma, this is simply equal to the uniform distribution on permutation sequences  $\pi_1, \dots, \pi_m$ , subject to

$$\prod_{j=1}^m \pi_j = \begin{cases} a & \text{if } B(x_1, \dots, x_n) = 1 \\ I & \text{if } B(x_1, \dots, x_n) = 0 \end{cases}.$$

This holds for any valuation of  $x_1, \dots, x_l$ . Now, the simulator's output, when unraveled in the same way as with the real Alice, also possesses the above

distribution, for any valuation of  $x_1, \dots, x_l$ . This follows immediately from Step 2 of the simulator. The lemma follows. ■

Thus, in terms of knowledge, a dishonest Bob had might as well select some  $x_1, \dots, x_l$  himself, and run the protocol. If he wasn't concerned with hiding his information, he might as well just tell Alice his  $x_1, \dots, x_l$ , and have her compute  $B(x_1, \dots, x_n)$  for him.

#### 4.3.4 Dealing with a malicious Alice.

In HONEST-W5PBP-EVAL, it is assumed that Alice behaves perfectly honestly. However, there are two ways in which Alice can be malicious.

1. She may try to steal information from Bob, finding out information about  $x_1, \dots, x_l$ .
2. She can try to get Bob to reconstruct some value of  $B(x_1, \dots, x_n)$  without herself deciding on some value of  $x_{l+1}, \dots, x_n$ , or simply reconstruct a value of  $B(x_1, \dots, x_n)$  which is wrong.

To address these problems, we would like to fix our protocol so that

1. Alice never gets any information about  $x_1, \dots, x_l$ .
2. If Bob reconstructs some value  $b$ , it must be the case that
  - (a) Given Alice's set of transmissions, one can uniquely compute a set of values  $x_{l+1}, \dots, x_n$ .
  - (b)  $b = B(x_1, \dots, x_n)$ .

This is actually quite easy to accomplish, given our commital and decommital technology. We first have Alice commit all the permutations she initially creates, along with proof that they correspond to a correct run of the protocol. All of the string transfers and transmissions will be oblivious decommitals and decommital of these committed bits. The protocol for oblivious W5PBP evaluation is given in Figure 4.7.

**Protocol W5PBP-Eval( $B, l, n, k$ )**

**1:** Alice uniformly chooses a randomizer  $R$ , and computes  $B^R$ . Alice then computes  $V_i(B^R)$  for  $i \in [1, n]$ . Alice then uses protocol SECURE-ZERO-KNOWLEDGE-COMMIT to commit

1.  $R$ ,
2.  $V_i(B^R)$ , for  $i \in [1, l]$ ,
3. Her choice of  $x_i$ , for  $i \in [l + 1, n]$ , and
4.  $V^{x_i}(B^R)$ , for  $i \in [l + 1, n]$ .

This commital is accompanied by a zero-knowledge proof that the  $V_i(B^R)$ 's and the  $V^{x_i}(B^R)$ 's are correctly computed, given  $R$  and  $x_{l+1}, \dots, x_n$ . This predicate can be made to be in  $NC^1$  through the judicious use of extra committed bits.

- 2:** Alice and Bob execute  $l$  parallel runs of OBLIVIOUS-DECOMMIT. For  $i \in [1, l]$ , Alice obliviously decommits either  $V_i^0(B^R)$  or  $V_i^1(B^R)$ .
- 3:** For  $i \in [l+1, n]$ , Alice runs SECURE-ZERO-KNOWLEDGE-DECOMMIT. Alice decommits  $V^{x_i}(B^R)$ . If Bob detects any inconsistency in reconstructing bits during Steps 2 and 3, he concludes that he cannot reconstruct  $B(x_1, \dots, x_n)$ . Otherwise, Bob uses his information to reconstruct  $B(x_1, \dots, x_n)$ .

Figure 4.7: Protocol for oblivious W5PBP evaluation (general case).

### Can Alice cheat Bob?

We wish to argue that in order for Bob to recover some value,  $b$ , a malicious Alice must essentially commit to values for  $x_{l+1}, \dots, x_n$ , and it must be the case that  $b = B(x_1, \dots, x_n)$ . This follows immediately, since Alice is forced to commit her arguments and all the permutations she uses throughout the protocol, along with a proof that they chosen according to the specifications of the protocol. After the commital phase is over, the game is all in Bob's hands. All Alice can do is to violate the oblivious decommittal protocols. But, with high probability, this will either not affect the game at all, or cause Bob to realize that Alice is cheating. Alice can't trick Bob into recovering different strings than what she committed.

It is also not hard to see that even a malicious Alice cannot learn anything about Bob's input. The only point where Bob talks to Alice is during the oblivious decommittal phase in Step 2, and Alice gets no information from this protocol. Note that Bob does not give Alice any feedback about whether he successfully reconstructed all the obliviously decommitted bits.

A somewhat uncomfortable point is that Bob cannot reveal that he did not receive a value. This suffices for the chief application of our thesis, but is unsatisfactory. One would like to allow Bob to complain, without fear of leaking information. At the end of the this chapter, we will present a simple scheme for handling this problem.

### Bounding Bob's knowledge.

We would like to be able to say that the new W5PBP evaluation protocol yields no more knowledge to Bob than the old one. To this end, we present a nearly identical simulator, given in Figure 4.8.

**Lemma 4.6** Let W5PBP  $B$  be as above, and let  $k \geq (nm)^c$ , for some  $c > 0$ . Then for any strategy Bob employs, the output of simulator  $\text{W5PBP-EVAL}(B, l, n, k)$  will be indistinguishable from the output of protocol  $\text{W5PBP-EVAL}(B, l, n, k)$ .

**Proof:** the proof is essentially the same as with the protocol for the honest case. All that has changed is that we have added a commital step, and changed calls to OST into calls to OBLIVIOUS-DECOMMIT. We know that these give the correct simulations, provided the Alice they are talking to is

**Simulator W5PBP-Evaluate( $B, l, n, k$ )**

- 1: The simulator runs, as a subsimulation, simulator SECURE-ZERO-KNOWLEDGE-COMMIT( $N, k, P$ ). Here,  $P$  is the same predicate as with the commital protocol, and  $N$  is the total number of bits committed by the protocol.
- 2: Let  $q_i$  be the length of  $\mathcal{V}_i^0(B)$  (note that this equals  $V_i^j(B^R)$ , for any randomizer  $R$ , and  $j \in \{0, 1\}$ ). For  $i \in [1, l]$ , the simulator executes, in parallel, subsimulator OBLIVIOUS-DECOMMIT. Simulator OBLIVIOUS-DECOMMIT will, in the course of its execution, make  $l$  queries to the simulated Alice, which we denote  $x_1, \dots, x_l$ . The simulator sends  $x_1, \dots, x_l$  to Alice, who returns  $B(x_1, \dots, x_n)$ .
- 3: The simulator uniformly chooses  $\pi_1, \dots, \pi_m$ , subject to

$$\prod_{i=1}^m \pi_i = \begin{cases} a & \text{If } B(x_1, \dots, x_n) = 1 \\ I & \text{If } B(x_1, \dots, x_n) = 0 \end{cases}.$$

For  $i \in [1, n]$  the simulator computes

$$V_i^{sim} = \pi_{var_1^i(B)}, \pi_{var_2^i(B)}, \dots.$$

- 4: The simulator gives the OBLIVIOUS-DECOMMIT subsimulator the values  $V_1^{sim}, \dots, V_l^{sim}$  as answers to its queries.
- 5: For  $i \in [l + 1, n]$ , the simulator makes the appropriate calls to subsimulator SECURE-ZERO-KNOWLEDGE-DECOMMIT to simulate the bitwise decommittal of the strings  $V^{x_i}(B^R)$ . The value of  $V^{x_i}(B^R)$  that is “decommitted” will be  $V_i^{sim}$ .

Figure 4.8: Simulator for oblivious W5PBP evaluation (general case).

correctly simulated. The simulation of Alice is the same as with the honest case, so this part of the lemma follows by same proof as before. ■

### 4.3.5 Functions with multi-bit outputs.

The W5PBP-EVAL protocol allows people to obviously evaluate  $NC^1$  circuits which have a single output bit. However, we would like to be able to evaluate circuits which have multi-bit outputs. We can accomplish this goal by representing each output bit of  $NC^1$  circuit as the output of a W5PBP. Naively, we would then run several independent versions of the W5PBP protocol, one for each W5PBP. This won't quite work, because we need to enforce the constraint that Bob's arguments to each branching program are the same. Fortunately, this constraint can be enforced by merging all the OBLIVIOUS-DECOMMIT protocols for a particular variable into a single OBLIVIOUS-DECOMMIT protocol. To this end, we make the following definition.

**Definition 4.3** Let  $\mathcal{B} = B_1, \dots, B_M$  be a sequence of W5PBP's with inputs  $x_1, \dots, x_n$ . We define  $\mathcal{V}_i(\mathcal{B}) = (\mathcal{V}_i^0(\mathcal{B}), \mathcal{V}_i^1(\mathcal{B}))$  by

$$\mathcal{V}_i^j(\mathcal{B}) = V_i^j(B_1), \dots, V_i^j(B_M)$$

Our protocol for general  $NC^1$  circuit evaluation is almost typographically the same as the W5PBP-EVAL protocol, which is as to be expected. Likewise, the simulator is nearly the same as the old simulator. We present the protocol and simulator in Figures 4.9 and 4.10, respectively.

As before, the only messages which Bob sends to Alice are in the parallel OBLIVIOUS-DECOMMIT phase. By the same argument as before, this gives Alice no information if Bob never complains.

Likewise, provided that  $k > (nmM)^c$ , for some  $c > 0$ , the indistinguishability proof of the single-bit case carries over to the multi-bit case. One might wonder why the bounds on  $k$  seem to grow progressively more complex. All we require is that  $k$  is sufficiently large in every subprotocol it is used in. In this case, the size of the all the sets of bits being manipulated is bounded above by some polynomial in  $m, n, M, k$ , so as long as  $k > (nmM)^c$ , for some  $c > 0$ , it will also be bounded above by some polynomial in  $k$ .

**Protocol  $NC^1\text{-Eval}(C, l, n, k)$**

1: Let  $\mathcal{B} = B_1, \dots, B_M$ , where each W5PBP  $B_i$  evaluates output bit  $i$  of  $C$ . Let each W5PBP  $B_i$  be of length  $m$  (we can assume that they are all of equal length, since we can pad out short W5PBP's with triples of the form  $(x_i, I, I)$ ). Alice uniformly chooses randomizers  $\mathcal{R} = R_1, \dots, R_M$ , and computes  $\mathcal{B}^{\mathcal{R}} = B_1^{R_1}, \dots, B_M^{R_M}$ . Alice then computes  $\mathcal{V}_i(\mathcal{B}^{\mathcal{R}})$  for  $i \in [1, n]$ . Then, using protocol **SECURE-ZERO-KNOWLEDGE-COMMIT**, Alice commits

1.  $\mathcal{R}$ ,
2.  $\mathcal{V}_i(\mathcal{B}^{\mathcal{R}})$ , for  $i \in [1, l]$ ,
3. Her choice of  $x_i$ , for  $i \in [l+1, n]$ , and
4.  $\mathcal{V}_i^{x_i}(\mathcal{B}^{\mathcal{R}})$ , for  $i \in [l+1, n]$ , and her choice of  $x_i$ .

This commital is accompanied by a zero-knowledge proof that the  $\mathcal{V}_i(\mathcal{B}^{\mathcal{R}})$ 's and the  $\mathcal{V}_i^{x_i}(\mathcal{B}^{\mathcal{R}})$ 's are correctly computed, given  $\mathcal{R}$  and  $x_{l+1}, \dots, x_n$ . This predicate can be made to be in  $NC^1$  through the judicious use of extra committed bits.

- 2: Alice and Bob execute  $l$  parallel runs of **OBLIVIOUS-DECOMMIT**. For  $i \in [1, l]$ , Alice obviously decommits either  $\mathcal{V}_i^0(\mathcal{B}^{\mathcal{R}})$  or  $\mathcal{V}_i^1(\mathcal{B}^{\mathcal{R}})$ .
- 3: For  $i \in [l+1, n]$ , Alice runs **SECURE-ZERO-KNOWLEDGE-DECOMMIT**. Alice decommits  $\mathcal{V}_i^{x_i}(\mathcal{B}^{\mathcal{R}})$ . If Bob detects any inconsistency in reconstructing bits during Steps 2 and 3, he concludes that he cannot reconstruct the values of  $B_i(x_1, \dots, x_n)$ , for  $i \in [1, M]$ . Otherwise, Bob uses his information to reconstruct  $B_i(x_1, \dots, x_n)$ , for  $i \in [1, M]$ .

Figure 4.9: Protocol for oblivious  $NC^1$  circuit evaluation.

**Simulator  $NC^1$ -Evaluate( $C, l, n, k$ )**

- 1: The simulator runs, as a subsimulation, simulator SECURE-ZERO-KNOWLEDGE-COMMIT( $N, k, P$ ). Here.  $P$  is the same predicate as with the commital protocol, and  $N$  is the total number of bits committed by the protocol.
- 2: Let  $\mathcal{B}$  be as defined in the protocol. Let  $q_i$  be the length of  $\mathcal{V}_i^0(\mathcal{B})$  (note that this equals  $V_i^j(\mathcal{B}^{\mathcal{R}})$ , for any randomizer  $\mathcal{R}$ , and  $j \in \{0, 1\}$ ). For  $i \in [1, l]$ , the simulator executes, in parallel, subsimulator OBLIVIOUS-DECOMMIT. Subsimulator OBLIVIOUS-DECOMMIT will, in the course of its execution, make  $l$  queries to the simulated Alice, which we denote  $x_1, \dots, x_l$ . The simulator sends  $x_1, \dots, x_l$  to Alice, who returns

$$B_1(x_1, \dots, x_n), \dots, B_M(x_1, \dots, x_n).$$

- 3: For  $i \in [1, m]$  and  $j \in [1, M]$  the simulator uniformly chooses  $\pi_i^j$ , subject to

$$\prod_{i=1}^m \pi_i^j = \begin{cases} a & \text{If } B_i(x_1, \dots, x_n) = 1 \\ I & \text{If } B_i(x_1, \dots, x_n) = 0 \end{cases}.$$

For  $i \in [1, n], j \in [1, M]$  the simulator computes

$$V_i^{sim}[j] = \pi_{var_1^i}^j(B_j), \pi_{var_2^i}^j(B_j), \dots.$$

The simulator then computes

$$\mathcal{V}_i^{sim} = V_i^{sim}[1], \dots, V_i^{sim}[M]$$

- 4: The simulator gives the OBLIVIOUS-DECOMMIT subsimulator the values  $\mathcal{V}_1^{sim}, \dots, \mathcal{V}_l^{sim}$  as answers to its queries.
- 5: For  $i \in [l + 1, n]$ , the simulator makes appropriate calls to the simulator for SECURE-ZERO-KNOWLEDGE-DECOMMIT, to simulate the bitwise decommital of  $\mathcal{V}^{x_i}(\mathcal{B}^{\mathcal{R}})$ . The value of  $\mathcal{V}^{x_i}(\mathcal{B}^{\mathcal{R}})$  that is “decommitted” will be  $\mathcal{V}_i^{sim}$ .

Figure 4.10: Simulator for oblivious  $NC^1$  circuit evaluation.

### 4.3.6 Compressing the number of rounds of communication.

The NC<sup>1</sup> circuit evaluation protocol clearly takes only a constant number of rounds of communication, since it consists of only a constant number of steps, each of which only require a constant number of rounds of communication. In fact, one can trivially “merge” the steps of this protocol into an essentially optimal number of rounds. In Step 1 of the protocol, Alice commits her bits by transferring a single message to Bob. In Step 2 of the protocol, Alice and Bob perform a set of oblivious decommitals in parallel. Oblivious decommitals are nothing more than glorified 1-2 oblivious string transfers, which consist of Alice transferring a string to Bob, Bob sending a message to Bob, and Alice sending a message back to Bob. In Step 3 of the protocol, Alice performs a set of decommitals, which just consists of sending a message to Bob.

Now, the transfer Alice executes at the beginning of Step 2 can be combined with the transfer she executes in Step 1. Her message to Bob at the end of Step 2 can be combined with the message she sends in Step 3 of the protocol. Thus, the final protocol can be written in the following form.

- Alice transfers a set of bits to Bob.
- Bob sends Alice a message.
- Alice sends Bob a message.

In general, this is the best that can be done, even when both parties are honest. There is no point in Bob sending any messages to Alice before she executes oblivious transfer: Since the message can’t have any correlation with Bob’s input (to maintain obliviousness), Alice could just as easily generate the message distribution herself. Similarly, one can argue that Bob must send a message to Alice, and get some reply. This is hardly a proof, and we leave the details to the reader.

It is an interesting open question whether general circuit evaluation can be accomplished using only a single transfer and two messages. In the cryptographic scenario, we have the constant-round protocols of [Y] and [GMW], but these techniques break down in the noncryptographic domain.

### 4.3.7 Allowing Bob to complain.

As far as the scope of this thesis is concerned, there is no need to allow Bob to complain if he gets a bad string in the one of the OBLIVIOUS-DECOMMIT protocols. However, from an aesthetic point of view, we would like to give some idea of how to improve our scheme. In fact, we do not need to modify the NC<sup>1</sup>-EVAL protocol, but merely change the way it is used.

Recall that Alice could not give Bob any decommittal string that would cause him to reconstruct any bits other than what was committed. However, she could give him strings which he would reject with probability at least 1/2. By knowing whether Bob received a bad decommittal string, Alice could get information about which strings he received, which in turn would yield information about Bob's bits.

Now suppose Bob randomized his actual arguments, yielding values  $x_1, \dots, x_l$  which have the following property. For any set of  $q$  indices,  $i_1, \dots, i_q$ , the values of  $x_{i_1}, \dots, x_{i_q}$  are uniformly distributed, regardless of what Bob's original values were. Then, the following two statements follow by elementary probability theory.

1. Suppose that Alice makes (intentional) mistakes in only  $r < q$  of the pairs of strings that she runs OST with in the OBLIVIOUS-DECOMMIT protocol. The probability of Bob complaining will be the same regardless of his original arguments.
2. Suppose that Alice makes (intentional) mistakes in at least  $q$  of the pairs of strings that she runs OST with in the OBLIVIOUS-DECOMMIT protocol. Then Bob will complain with probability at least

$$1 - \left(\frac{3}{4}\right)^{q-1}.$$

The first fact follows from the randomization property of Bob's arguments. The second property also subtly follows from the randomization property. Any bad pair will be detected by Bob with probability at least 1/4, since a bad string will be chosen by Bob with probability 1/2, and this bad string will be detected with probability at least 1/2. For up to  $q - 1$  pairs, these 1/4 probability will be independent of each other, giving the desired bound.

Thus, if  $q$  is large, say equal to  $k$ , then Alice will learn nothing from cheating, even if Bob complains. If she deviates on no more than  $q$  pairs,

then Bob's probability of complaining will not depend on his original set of arguments. If she deviates on  $q$  pairs, then Bob will almost always complain.

The only question that remains, then, is how can Bob randomize his original arguments to have the  $(q - 1)$ -independence property. The trick is to use the standard parity trick. Given some circuit  $C(x_1, \dots, x_n)$ , we define the  $nq$  argument circuit,  $C_q(x_{1_1}, \dots, x_{n,q})$  by

$$C_q(x_{1_1}, \dots, x_{n,q}) = C(x_1, \dots, x_n),$$

where,

$$x_i = \bigoplus_{j=1}^q x_{i,j}.$$

Clearly, if  $C$  is an  $NC^1$  circuit,  $C_q$  will also be  $NC^1$ .

To evaluate  $C(x_1, \dots, x_n)$ , with complaining, Alice and Bob just run  $NC^1$ -EVAL on  $C_q(x_{1_1}, \dots, x_{n,q})$ , where Bob uniformly chooses his arguments subject to the above parity constraint. This randomization satisfies the  $(q - 1)$ -independence property, so we are done.

## Chapter 5

# Oblivious Evaluation of Arbitrary Circuits.

In the last chapter, we developed protocols and simulators for the secure evaluation of  $NC^1$  circuits. In this chapter, we show how to extend these techniques to the secure evaluation of arbitrary polynomial-sized circuits. For the rest of our discussion, we will express all our protocols in terms of  $NC^1$  circuit evaluation, and not resort to any lower-level primitives. This discipline will make our arguments much less agonizing.

In order to use our  $NC^1$  circuit capability, we develop some simple machinery which will allow us to further constrain Alice and Bob's behavior. We refer to the first technique we use as the *method of encrypting conversations*.<sup>1</sup> This technique has a two-fold purpose. Using it we can hide from Bob the intermediate values computed in a series of  $NC^1$  circuit evaluations. We can also use this technique in order to force Bob to behave in an honest fashion. The second technique, known as *consistency checking*, allows us to have a great deal of control over Alice's input to a series of circuits. Consistency checking, a standard technique in cryptography, provides a mechanism through which Bob can be convinced that Alice's input to a given circuit is the same as her input to another circuit.

---

<sup>1</sup>This technique was concurrently and independently discovered by Chaum, Damgård, and Van de Graaf [CDG], and slightly simpler form of it was also independently discovered by Goldreich and Vainish [GV]. Given its naturalness, it has probably been discovered by many others as well. However, we are the first to give it a snazzy name.

## 5.1 General secure circuit computation. Reducing to the $\text{NC}^1$ case.

The result we have been ultimately aiming for, and which subsumes all the previous results of this investigation, is a reduction from secure circuit computation to oblivious transfer. The idea of our reduction is very simple. Any circuit can be thought of as a series of shallow circuits chained together. Since we can chain together  $\text{NC}^1$  circuits, we can securely evaluate arbitrary circuits. We expand on this idea below.

Without loss of generality, we assume that all the circuits we are dealing with are levelled. By a levelled circuit, we mean a circuit in which each gate may be assigned an integer value, denoting its level, subject to the following constraints.

1. All gates which are directly connected to the input have value 1.
2. If the output of gate  $a$  is piped into the input of gate  $b$ , then the level of  $b$  is one greater than the level of  $a$ .

We denote the depth of a levelled circuit by the maximum level of its gates. Clearly, we can take a nonlevelled circuit, and construct an equivalent levelled circuit whose size is polynomial in the size of the original circuit.

We can now state our problem as follows. Let  $C(x, y)$ , where  $x$  and  $y$  are  $n$  bit inputs, be a levelled circuit of size  $s(n)$  and depth  $d(n)$ .<sup>2</sup> We assume that  $s$  is polynomial in  $n$ . Bob knows  $x$  and Alice knows  $y$ . Alice and Bob wish to securely evaluate  $C$  using an oblivious transfer channel. Bob is allowed to learn  $C(x, y)$ , and nothing else, and Alice is to learn nothing. We give a pictorial description of this process in Figure 5.1.

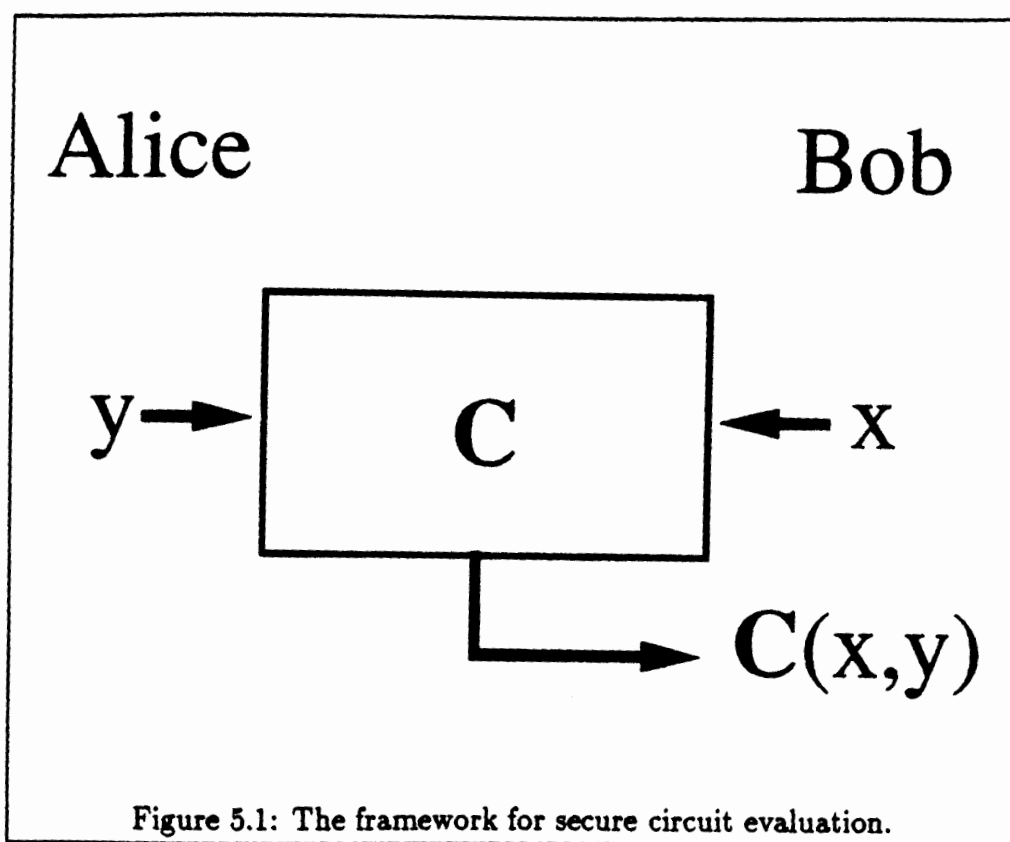
We can break up  $C$  into  $m = \lceil d(n)/\log n \rceil$  circuits,  $C_1, \dots, C_m$ , with the following properties.

1. For  $i \in [1, m]$ , circuit  $C_i$  has depth  $O(\log n)$ .
2. The following identity holds:

$$C(x, y) = C_m(C_{m-1}(\dots C_1(x, y) \dots))$$

---

<sup>2</sup>Following the usual convention, we allow  $n$  to be a variable, in which case we are implicitly denoting a circuit family.



This decomposition is straightforward. First, we define  $C_i$  as the circuit obtained by copying levels  $(i - 1) \log n + 1, \dots, \max(d(n), i \log n)$  of  $C$ . The inputs to  $C_i$  simply correspond to the inputs of the level  $(i - 1) \log n + 1$  gates, and the outputs of  $C_i$  are the outputs of the level  $\max(d(n), i \log n)$  gates. Such a decomposition meets the above conditions. Essentially, the input to  $C$  is fed into circuit  $C_1$ , the output of circuit  $C_i$  is piped into the input of circuit  $C_{i+1}$ , and the output of  $C_m$  corresponds to the output of  $C$ . We give a pictorial description of the decomposition in Figure 5.2.

The decomposition described above shows how to reduce the evaluation of general circuits to the evaluation of  $\text{NC}^1$  circuits, but does not show how to reduce the secure evaluation of general circuits to the secure evaluation of  $\text{NC}^1$  circuits. Naively, one might securely evaluate circuits  $C_1, \dots, C_m$ , piping the output of each circuit into the next, but such a system would be insecure in two distinct ways:

1. Bob is supposed to make his input to circuit  $C_{i+1}$  equal to the output of circuit  $C_i$ . However, a malicious Bob may violate this constraint.
2. Bob receives all of the intermediate outputs of  $C_1, \dots, C_{m-1}$  as well as the final output of  $C_m$ .

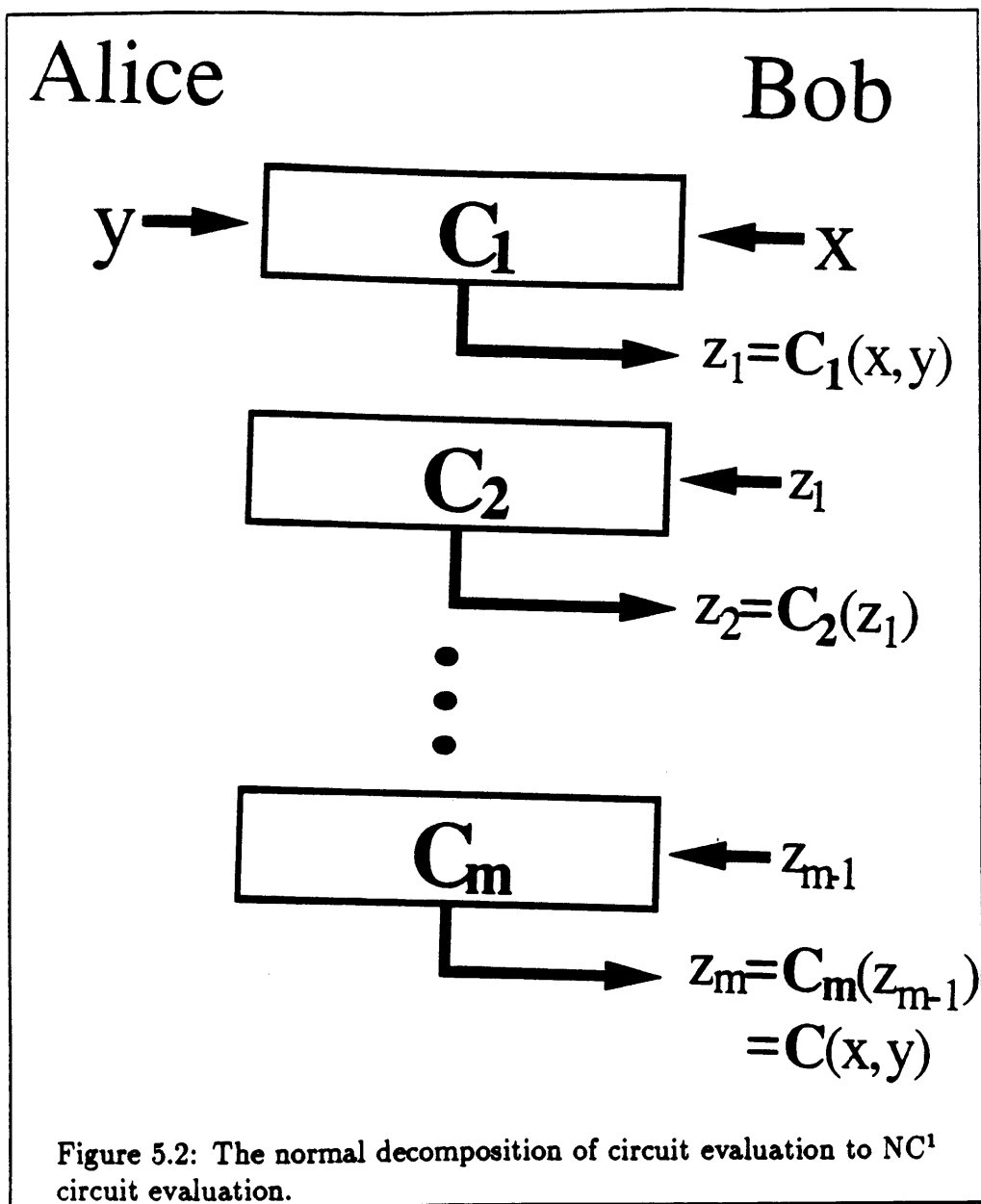
We must deal with both these ways in which the naive protocol may leak knowledge. To do this, we use the method of encrypted conversations. In order to implement this method, it is necessary to enforce consistency constraints on the behavior of Alice, which we show how to do at the end of the chapter.

Our end result is a protocol for chaining together  $\text{NC}^1$  circuits  $C_1, \dots, C_m$  in the manner described above, without giving Bob knowledge about any of the intermediate computations. We call this protocol  $\text{CHAIN}(C_1, \dots, C_m, k)$ .<sup>3</sup> To securely evaluate a circuit  $C$ , Alice and Bob can simply decompose  $C$  into  $\text{NC}^1$  circuits  $C_1, \dots, C_m$ , and chain them together using the  $\text{CHAIN}$  protocol. We give the top-level protocol, and its simulator, in Figures 5.3 and 5.4.

As is clear, the real work to be done is in the implementation of the  $\text{CHAIN}$  protocol.

---

<sup>3</sup>As usual,  $k$  is a security parameter.



**Protocol Circuit-Eval( $C, k$ )**

- 1: Let  $C$  be decomposed into  $C_1, \dots, C_m$ . Alice and Bob run protocol  $\text{CHAIN}(C_1, \dots, C_m, k)$ .

Figure 5.3: Top level protocol for secure circuit evaluation.

**Simulator Circuit-Eval( $C, k$ )**

- 1: Let  $C$  be decomposed into  $C_1, \dots, C_m$ . The simulator runs sub-simulator  $\text{CHAIN}(C_1, \dots, C_m, ks(n), k)$ .

Figure 5.4: Top level simulator for secure circuit evaluation.

## 5.2 Encrypted Conversations.

In this section, we describe the main idea behind our protocol for securely chaining together a series of  $\text{NC}^1$  circuits. We consider the following scenario. Alice and Bob have a sequence  $C_1, \dots, C_m$  of  $\text{NC}^1$  circuits they wish to evaluate. We assume that each circuit takes  $n$  bit inputs from Alice and Bob, and gives an  $n$ -bit output. This is not a serious restriction, since we can add dummy inputs and outputs to the circuit, and make  $n$  be equal to the maximum number of bits that any of our circuits will need.

We want to enforce constraints on Bob's knowledge and freedom of action that are much more stringent than those we have enforced so far. These are

1. For  $i \in [1, m - 1]$ , the output of  $C_i$  is to be used as Bob's input to circuit  $C_{i+1}$ . Otherwise, Bob is to gain no information.
2. For  $i \in [1, m - 1]$ , Bob receives no information about the output of circuit  $C_i$ . He does receive the output of  $C_m$ , provide he obeys the first condition.

The above two conditions are at first glance paradoxical. We want Bob to carefully chain the output of one circuit into the input of the next, but we do not want him to know these values. It is like asking a courier to deliver a message, but not wanting him to read it en route. To see the main idea

of how this paradox is resolved, we give a solution based on a physical abstraction we call secure envelopes.

### 5.2.1 A solution using secure envelopes.

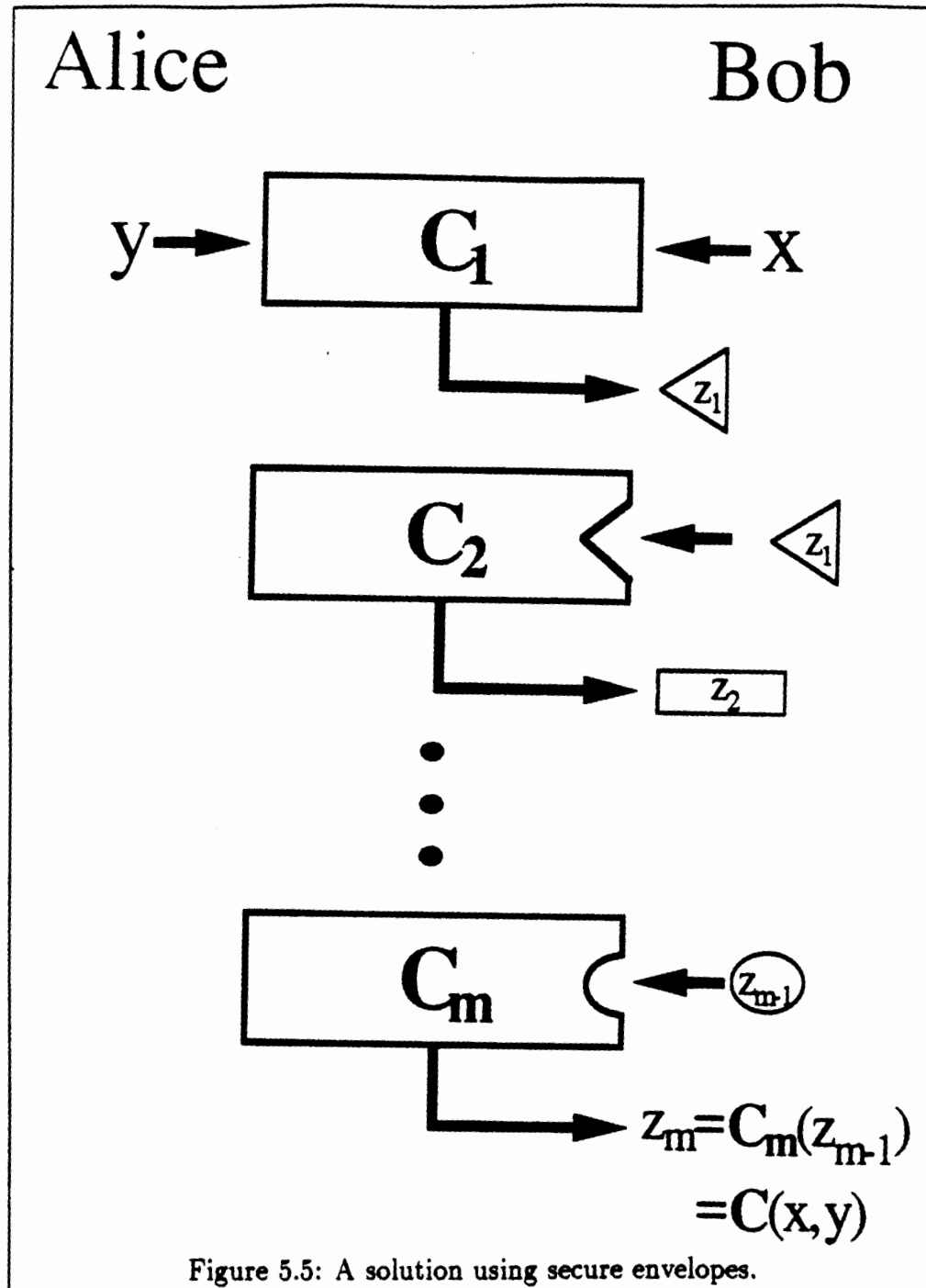
To give some intuition on how our reduction works, imagine that we have black boxes which implement secure  $\text{NC}^1$  circuit evaluation. These boxes can be programmed to, instead of working with numbers in the clear, take as input and give as output an abstraction known as *secure envelopes*. A secure envelope can only be created or opened by a black box; its contents are invisible to Bob, and cannot be altered in any way. Secure envelopes can also be made in different shapes, and the black boxes can be programmed to create or accept envelopes of a particular shape.

We now give a solution in this scenario, which is depicted in Figure 5.5. Alice and Bob make a series of black boxes which securely evaluate circuits  $C_1, \dots, C_m$ . All of these black boxes take ordinary inputs from Alice. The black box which evaluates  $C_1$  takes an ordinary input from Bob, and the box which evaluates  $C_m$  gives an ordinary output to Bob. All of the other inputs and outputs are put into secure envelopes. The box evaluating  $C_i$  puts its answer in a secure envelope which is of a shape only accepted by  $C_{i+1}$ . If a box receives an input which is of the wrong shape, then it will refuse to give out an envelope of any type.

We wish to informally argue that this scheme blocks off the two ways the naive protocol can leak information. First, Bob gets no information about the intermediate computations, since the contents of the envelopes are invisible to him. Second, Bob is constrained to chain the output of  $C_{i-1}$  into the input of  $C_i$ . He cannot make any new envelopes, and if he ever gives an improper envelope to a box, he will not get a new envelope for the next box, and thus never get any more envelopes at all. Thus, if he attempts to maliciously cheat, he will receive nothing more than a collection of unreadable envelopes for his troubles.

### 5.2.2 Implementing secure envelopes with secure $\text{NC}^1$ circuit evaluation.

We now show how to simulate the existence of secure envelopes using our ability to perform secure  $\text{NC}^1$  computations. Essentially, our solution uses



an encryption scheme that provably has strong properties in the information theoretic setting. This may seem odd, since we do not know how to prove the security of most cryptosystems. However, for our purposes, we can have the number of bits specifying the code be longer than the message itself, in which case the problem is not difficult.

Specifically, we construct, for all  $n$ , encryption functions,  $K(x)$ , where  $x$  is of size at most  $n$ . These functions have the following three properties.

1. (invertibility) For all keys,  $K$ , and  $x, x'$  of size  $n$ ,  $K(x) \neq K(x')$  if  $x \neq x'$ .
2. (unbreakability) For any  $x, x'$  of size  $n$ , and  $K$  distributed uniformly over all such functions, the induced distribution on  $K(x)$  will be indistinguishable from the induced distribution on  $K(x')$ .
3. (unforgeability) Suppose that  $K$  has a uniform *a priori* distribution on  $K$ , and that  $K(x) = y$ , where  $x$  is an arbitrary message. Then for any  $z \neq y$ , the conditional probability that,  $(\exists x', |x| \leq n) K(x') = z$ , is at most  $2^{-n}$ .

The following lemma exhibits such an encoding scheme.

**Lemma 5.1** Fix  $n$ , and let  $p$  be a  $2n + 1$  bit prime. Define the set of keys,  $K$ , to be the set of all ordered pairs of the form  $(a, b)$ , where  $a, b \in Z_p$ , and  $a \neq 0$ . Define

$$K(x) = ax + b \bmod p.$$

The resulting encoding scheme is invertible, unbreakable, and unforgeable.

**Proof:** To prove invertibility, we note that if  $K = (a, b)$ , then

$$\begin{aligned} K(x) - K(x') \bmod p &= a(x - x') \bmod p \\ &\neq 0 \bmod p, \end{aligned}$$

for  $a, x - x' \neq 0 \bmod p$ . To prove unbreakability, we note that for any  $x, y$ , and  $K = (a, b)$ ,  $K(x) = y$  iff  $b = y - ax \bmod p$ . Thus, there are exactly  $p - 1$  values of  $K$  (one for each possible value of  $a$ ) such that  $K(x) = y$ . Hence, the induced distribution on  $K(x)$ , where  $x$  is arbitrary, and  $K$  is distributed uniformly, is equal to the uniform distribution on  $Z_p$ . To prove unforgeability, we note that if  $K$  has a uniform *a priori* distribution, and  $K(x) = y$ , then

the conditional distribution on  $K$  is equal to the distribution on  $(a, y - ax)$  (here, arithmetic is over  $Z_p$ ), where  $a$  is distributed uniformly over  $Z_p - \{0\}$ . Thus, for any fixed  $z$ , the conditional distribution on  $K^{-1}(z)$  will be uniform over all elements of the form

$$a^{-1}(z - (y - ax)) = x + a^{-1}(z - y) \bmod p.$$

For  $z \neq y$ , this corresponds to  $p - 1 \geq 2^{2n}$  distinct values, of which at most  $2^n$  values correspond to  $n$  bit integers. ■

### Incorporating the encoding/decoding scheme into $NC^1$ circuits.

Given  $NC^1$  circuit  $C_i(y, z_{i-1})$ ,<sup>4</sup> we wish to modify it so as to compute with properly encrypted inputs, “reject” forged inputs, and produce encrypted outputs. This new circuit will take an encrypted argument,  $e_{i-1}$ , from Bob, and produce for him an encrypted argument,  $e_i$ . The encryption process will be controlled by Alice.

To perform this transformation, we first note that the encoding scheme described above is easy to compute and invert (in a limited sense) using  $NC^1$  circuits. For a fixed value of  $p$ , and we can create an  $NC^1$  circuit that, on input  $K = (a, b)$  from Alice, computes  $K(x) = ax + b \bmod p$ . To see that we can perform inverses, note that our set of encryption functions is closed under inverse. If  $K = (a, b)$ , then its inverse,  $K^{-1}$ , is equal to  $(a^{-1}, -a^{-1}b)$ , where arithmetic is done over  $Z_p$ . Thus, an encryption circuit can double as a decryption circuit if Alice gives the inverse of the key she wants to decrypt under.

We now incorporate the encryption scheme into the original circuits. For simplicity, let us assume that circuit  $C_i$ ’s input from Bob and output to Bob is exactly  $n$  bits,<sup>5</sup> and that  $p$  is a  $2n + 1$  bit prime. We can construct a modified  $NC^1$  circuit,  $C_i[p]$ , which takes from Bob a  $(2n + 1)$ -bit encoded input, which we refer to as  $e_{i-1}$ . From Alice,  $C_i[p]$  takes the same input,  $y$ , that was taken by  $C_i$ , and two additional inputs, which we denote by  $K_{i-1}^{-1}$  and  $K_i$ . These additional inputs are encryption functions for the scheme described

<sup>4</sup>Note that, for  $i > 1$ , Alice will not actually input any values to circuit  $C_i$ , so the  $y$  is ignored. We include it as an argument to make our definitions cover all the cases.

<sup>5</sup>We can “pad” inputs and outputs by ignoring extraneous input pins and writing 0’s on extraneous output pins.

**Circuit  $C_i[p](y, e_{i-1}, K_{i-1}^{-1}, K_i)$**

- 1:** Treat  $e_{i-1}$  as a  $2n+1$  bit number. If  $z_{i-1} \geq p$ , then halt and output null ( $2n+1$  0's).
- 2:** Compute  $z_{i-1} = K_{i-1}^{-1}(e_{i-1})$ . If  $z_{i-1}$  has more than  $n$  bits (i.e.,  $z_{i-1} \geq 2^n$  when viewed as a number), then halt and output null.
- 3:** Compute  $z_i = C_i(y, z_{i-1})$ . Treating  $z_i$  as an  $n$ -bit number, compute  $e_i = K_i(z_i)$ , and output  $e_i$ .

Figure 5.6: Construction of  $NC^1$  circuit  $C_i[p]$ .

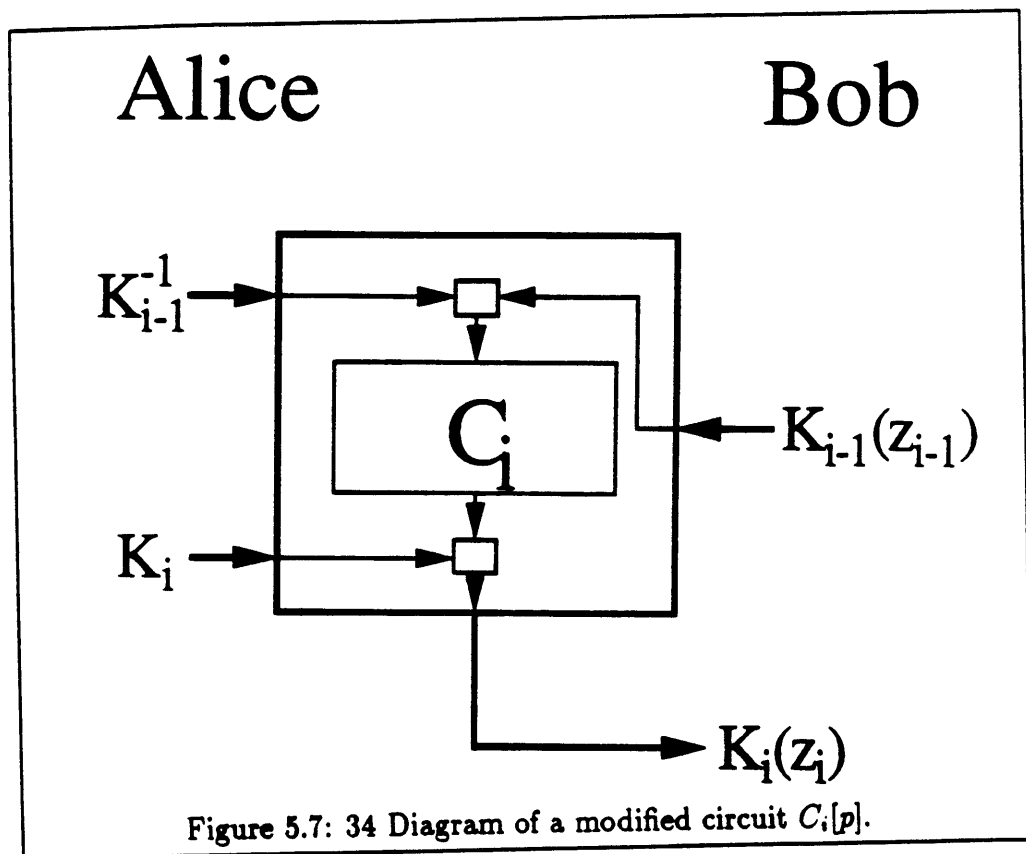
above, and are thus both represented by elements of  $Z_p \times Z_p$ . The meanings of the subscripts and inverses will be more apparent from the description of the CHAIN protocol. A more formal specification of this transformation is given in Figure 5.6, and a transformed version of circuit  $C_i$  is depicted in Figure 5.7.

We note that this transformation can be made to cover circuits  $C_1$  and  $C_m$ . The  $y$  input is handled by the definition. The fact that  $C_1$  accepts unencrypted input from Bob, and  $C_m$  gives unencrypted output to Bob, can be preserved by defining  $K_0^{-1} = K_m = (1, 0)$ , the identity encryption. The other values of  $K_i$  will be selected at random in the larger protocol.

### 5.3 Implementing the chain protocol, assuming constraints on Alice's inputs.

Using our information theoretically secure encryption scheme, Alice can implement secure envelopes, and thus hide all of the intermediate results from Bob. This allows us to give a partial solution, based on what may seem at first to be a very large assumption. We assume that Bob can enforce arbitrary constraints on Alice's input to a set of  $NC^1$  circuits. We give the resulting protocol in Figure 5.8. In the next section, we show how to achieve such constraints.

Informally, this protocol prevents Bob from cheating, since all the additional information he receives is a series of encrypted intermediate computa-



**Protocol Naive-Chain( $C_1, \dots, C_m, k$ )**

- 1: Adding dummy inputs and outputs if necessary, ensure that for some  $n > k$ ,  $C_1, \dots, C_m$  give and receive  $n$  bit quantities to and from Bob. Alice and Bob agree on a  $2n + 1$  bit prime,  $p$ . This implicitly defines circuits  $C_1[p], \dots, C_m[p]$ . For convenience, we denote  $C_i[p]$  by  $C_i^*$ . Using modulus  $p$ , Alice uniformly chooses keys  $K_1, \dots, K_{m-1}$ , and defines  $K_0, K_m = (1, 0)$ .
- 2: Alice and Bob successively evaluate circuits  $C_1^*, \dots, C_m^*$ , with security parameter  $k$ , subject to a constraint to be described later. Bob gives  $C_1^*$  the input he would have given to  $C_1$ , plus  $n + 1$  0's put in for padding (since his  $n$  bit input is treated as a  $2n + 1$  bit number). His input to  $C_i^*$  will be the output of  $C_{i-1}^*$ . Alice's inputs to  $C_i^*$  are what she would have input to  $C_i$ , and keys  $K_{i-1}^{-1}, K_i$ . The following consistency constraints are enforced:
  1. If  $K_{in}^{-1}[i]$  or  $K_{out}[i]$  are of the form  $(a, b)$ , then  $a \in Z_p^*$  and  $b \in Z_p$ . That is, all keys are legal.
  2.  $K_{in}^{-1}[0] = K_{out}[m] = (1, 0)$
  3. If  $K_{out}[i-1] = (a, b)$ , and  $K_{in}^{-1}[i] = (c, d)$ , then  $ac = 1 \bmod p$  and  $cb + d = 0 \bmod p$ . In other words, the keys are indeed inverse to each other.

If Bob sees a violation of these constraints, he concludes that the final output of circuit  $C_m^*$  is meaningless, otherwise he recovers the value that  $C_m$  would have had, which will be equal to the value of  $C_m^*$ , up to padding bits (since the  $N$  bit output is represented by a  $2n + 1$  bit number).

Figure 5.8: A naive protocol for chaining together NC<sup>1</sup> circuits.

tions. If Alice gives the encryption functions specified by the protocol, they will not affect the final answer; the encryption of  $C_i[p]$ 's output is reversed by circuit  $C_{i+1}[p]$ . However, since the protocol assumes that Bob can tell whether Alice gave reasonable inputs, it is still part wishful thinking. For this reason, we do not give any detailed analysis.

## 5.4 Consistency Checking.

We consider the following scenario. Alice and Bob wish to securely evaluate a set of  $\text{NC}^1$  circuits,  $C_1, \dots, C_n$ . If they evaluate these circuits in the usual fashion, i.e. independently, then both Alice and Bob can make their inputs to each circuit independent from their inputs to any of the other circuits. However, may may wish to impose some constraints on Alice, of the form, "bit 6 of Alice's input to circuit 5 must be equal to bit 7 of circuit 17." Essentially, we would like to enforce consistency constraints on Alice's inputs to her circuits.

Up to now, our methodology has not addressed this issue. However, consistency checking is a common problem with many simple solutions. Given the ability to securely evaluate  $\text{NC}^1$  circuits, it is particularly easy. The following solution is based largely on a suggestion by Crépeau.

First, we use an expanded representation for bits. Given a bit  $b$ , and security parameter  $k$ , we represent  $b$  by an  $3k$  by 3 matrix,  $B = [b_{i,j}]$ , such that the exclusive-or of the bits on any row of  $B$  is equal to  $b$ . That is,

$$\forall i \in [1, mk]) \quad b = \bigoplus_{j=1}^3 b_{i,j}.$$

We call such a matrix an *k-expansion* for  $b$ . For example, here is a  $k$ -expansion for 1.

$$\begin{array}{ccc} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{array}$$

We note the following two simple facts:

1. If  $B_0$  and  $B_1$  are  $k$ -expansions for 0 and 1, respectively, then  $B_0$  and  $B_1$  must differ in at least  $k$  locations.
2. Let  $b$  be a bit chosen according to some *a priori* distribution, and let  $B$  be a uniformly chosen  $k$ -expansion for  $b$ . Revealing any 2 elements from each of the rows of  $B$  will not alter the conditional distribution on  $b$  (i.e. this information gives no information about  $b$ ).

These both follow from the properties of the parity function.

Suppose we have an  $\text{NC}^1$  circuit,  $C(y, x)$ , in which Bob has input  $x$ , and Alice has input  $y = y_1, \dots, y_l$  (i.e. an  $n$  bit input). We can create a modified  $\text{NC}^1$  circuit  $C\{k\}$ , with the following properties.

- $C\{k\}$  has Bob's normal input  $x$ , i.e. the input taken by  $C$ . However, instead of having input bits,  $y = y_1, \dots, y_n$ , Alice has input

$$Y = Y_1, \dots, Y_n,$$

where each  $Y_i$  corresponds to a  $k$ -expansion for  $y_i$ .

- Bob has additional inputs,  $c_j^i \in [1, 3]$  for  $i \in [1, n]$  and  $j \in [1, 3k]$ .
- The output of  $C\{k\}$  has outputs of three types. These are.
  1.  $C\{k\}$  outputs a "legitimacy" bit,  $Q$ , which is 1 iff  $Y_1, \dots, Y_n$  are legal  $k$ -expansions. If  $Q$  is equal to 0, then all of the other outputs will be all 0's.
  2. If  $Q = 1$ , then  $C\{k\}$  outputs the value of  $C(y, x)$ , where  $y = y_1, \dots, y_n$  and  $Y_i$  is a  $k$ -expansion for  $y_i$ .
  3. If  $Q = 1$ , then  $C\{k\}$  outputs a set of "checksum" bits, for  $i \in [1, n]$ , and  $q \in [1, 3k]$ . If we write  $Y_i = \{y_{q,j}^i\}$ , where  $q \in [1, 3k]$ , and  $j \in [1, 3]$ , then  $C\{k\}$  outputs  $y_{q,c_q^i}^i$ .

Intuitively, this scheme may be thought of as follows: Each of Alice's bits is represented by a  $3k \times 3$  matrix. During the circuit computation, Bob is allowed to see one element from each of the vectors. We now show how the above scheme can be used to enforce consistency constraints on Alice's inputs, while otherwise preserving her privacy. Here is a partial example for the checksum queries and outputs. If  $Y_1$  were the  $k$ -expansion given in the above example, and  $c_1^1, \dots, c_6^1$  were equal to 1, 3, 2, 3, 1, 2, then the circuits output would include 1, 0, 0, 0, 0, 1.

## Enforcing consistency constraints.

Suppose that Alice and Bob wish to evaluate circuits  $C_1, \dots, C_n$ , and Alice wants to convince Bob that some of her input bits are equal to others. We assume that each bit appears in only one consistency assertion, a restriction which will turn out to be unimportant. This consistency condition can be ensured by evaluating a set of modified circuits  $C_1\{k\}, \dots, C_n\{k\}$ , for appropriately chosen values of  $k$ . For each of her input bits of the original circuits, Alice gives a corresponding  $k$ -expansion. She chooses her  $k$ -expansions in the following manner.

- If Alice wishes to convince Bob that bit  $i$  of her input to  $C_a$  is equal to bit  $j$  of her input to  $C_b$ , then she uses the same  $k$ -expansion to represent both these input bits.
- Alice's  $k$ -expansions for an input bit are chosen uniformly from all legal expansions with that value, subject to the previous constraint.

Bob chooses his “checksum” inputs in the following manner.

- If Alice wishes to convince Bob that bit  $i$  of her input to  $C_a$  is equal to bit  $j$  of her input to  $C_b$ , then Bob uses the same set of checksum inputs on both these inputs. That is, the values of  $c_1^i, \dots, c_{mk}^i$  in circuit  $C_a$  will be equal to the values of  $c_1^j, \dots, c_{mk}^j$  in circuit  $C_b$ .
- Bob's choice of checksum inputs is uniform subject to the above constraint.

Now, suppose that Alice attempts to break one of the consistency constraints being imposed. If any of her  $k$ -expansions are invalid, then Bob will detect this fact immediately, since the legitimacy output of the circuit will be 0. However, if two valid  $k$ -expansions represent different bits, they must differ in at least  $3k$  places. However, since Bob sees one element of each of the vectors, chosen at random, he will note a discrepancy with probability

$$1 - \left(\frac{2}{3}\right)^{3k} \geq 1 - 2^{-\theta(k)}.$$

Thus, if  $k$  is our security parameter, then Alice cannot violate any consistency constraint with any significant chance of escaping detection.

## Does this scheme leak information to Bob?

*A priori*, allowing Bob to learn many of the bits in the  $k$ -expansions of Alice's input bits might compromise security. We now argue that if  $m$  is sufficiently large, this will not happen. The bound on the size of  $m$  depends on the set of consistency constraints imposed on Alice's inputs.

Every time a particular  $k$ -expansion,  $Y_i$ , is used, Bob is allowed to see exactly one element of his own choosing from each of the  $3k$  vectors which make  $X$  up. We have placed the restriction that a given bit can appear in at most one assertion, hence an expansion will appear at most twice. Hence, Bob will be allowed to see at most 2 elements from each of the rows of  $Y_i$ . However, since  $Y_i$  is uniformly chosen, this imparts no information about whether  $Y_i$  is an expansion of a 0 or a 1.

## Proving more general assertions about Alice's arguments.

Being able to prove assertions of equality between pairs of inputs allows one to also prove much more complicated assertions. For example, suppose Alice wants to convince Bob that her input  $d$  to circuit  $C_1$  is a factor of her input  $n$  to circuit  $C_2$ . All Alice and Bob need do is agree on a checking circuit,  $\text{CHECK}(a, b, c)$ , that takes inputs  $a, b, c$  from Alice, and return 1 iff  $c = ab$ . Alice and Bob can then evaluate  $C_1, C_2$ , and  $\text{CHECK}$ , enforcing the consistency constraint that the first and third inputs to  $\text{CHECK}$  are equal to the inputs  $d$  and  $n$  to  $C_1$  and  $C_2$ , respectively.

Similarly, Alice can prove any NP assertion about her inputs, provided she has a witness for the assertion.<sup>6</sup> Each of Alice's input bits are ensured via consistency checking to be duplicated in a final "checking" circuit. The constraint circuit takes all the inputs of the previously evaluated circuits, whatever further inputs Alice wishes to give it (i.e. witness bits), and outputs a 1 iff some predicate holds for these input bits. By agreeing on the checking circuit they use, Alice and Bob can enforce any NP assertion about Alice's inputs.

Thus, consistency checking allows us to perform commital with zero-knowledge proofs in a very flexible manner. One can commit inputs, use them

---

<sup>6</sup>We implicitly use the fact that any NP assertion can be verified by an  $\text{NC}^1$  circuit, given a good witness.

in circuit evaluations, and then at a much later time give a zero-knowledge proof for some assertion about the inputs. One need not even decide on what assertion to prove until this much later time. This flexibility is much greater than that afforded by the ordinary zero-knowledge commit protocols.

There is a price for all this flexibility. By using  $\text{NC}^1$  evaluation for all ones base operations, one might use slightly more rounds than if one “codes up” ones protocols from scratch. However, our  $\text{NC}^1$  protocols are extremely cheap, consisting of only three rounds.<sup>7</sup> More to the point, the number of rounds used in our general protocol will be not even close to what we can show is optimal, so there is no point quibbling over multiplicative constants.

### **Simulating the evaluation of $\text{NC}^1$ circuits with consistency constraints.**

We have seen how to evaluate a series of  $\text{NC}^1$  circuits with consistency constraints, and some of the ways this capability may be used. We would like to simulate this sequence of evaluations. This simulation is very straightforward, since we are still evaluating a sequence of ordinary, if slightly transformed  $\text{NC}^1$  circuits. Thus, we can use the simulator for ordinary  $\text{NC}^1$  circuit evaluation, provided we can simulate all of the outputs of these circuits. In addition to simulating the usual circuit outputs, which we get from a “friendly Alice” (or, as is often the case, a higher level simulator), we must also simulate the legitimacy output bit, and the checksum output bits. If Alice abides by the protocol, then the legitimacy output bit will always be equal to 1. The checksum bits output will be distributed uniformly, subject to the constraint that if the same bit of what should be two identical  $k$  expansions (according to the consistency constraint) are requested, then the same value will be returned in both instances. Thus, the extra outputs can be trivially simulated.

While this scheme is very simple, it is cumbersome to constantly write down all of the checksum queries, and explicitly expand out bits into  $k$ -expansions. In order to make our future protocols more readable, we will suppress the details of the consistency checking scheme, and its simulation, as much as possible. It should be emphasized that behind this somewhat

---

<sup>7</sup>In fact, only two rounds are needed in slightly modified scenarios, to be obsessive about the matter.

loose exposition is a very mechanical procedure for using and simulating our scheme.

Figure 5.9 gives our final protocol, in somewhat gory detail.

## Properties of the chain protocol.

The CHAIN protocol we describe has four important properties.

1. If Alice and Bob behave according to the protocol, then at its conclusion Bob gets an answer which is the correct result of chaining  $C_1, \dots, C_m$  together. More precisely, Alice's messages will implicitly define some inputs,  $Y$ , and Bob's messages will implicitly define some input,  $X$ . Bob will recover

$$C_m(C_{m-1}(\dots(C_2(C_1(X, Y))\dots))).$$

2. If Bob is honest, then no matter how Alice behaves, Bob will either get the correct result of chaining, or get no answer whatsoever, and Alice will learn nothing about Bob's inputs. More colloquially, Alice must either play or quit. She cannot cause Bob to recover garbage.
3. Alice learns nothing about Bob's inputs.
4. No matter what strategy Bob employs, his view of the conversation can be simulated by a simulator which is allowed to give an input,  $X$ , to a "friendly Alice," who will return the end result of chaining the circuits with her inputs.

The first three properties are relatively straightforward. The first property holds because all of the linear transformations over  $GF(p)$  which are applied to the outputs of the circuits are inverted before the next circuit computes with them. Thus, in terms of the final answer, it is as if the transformations were never employed. The second property holds because Alice is constrained to behave honestly by the commitment and consistency checking schemes. She has to give a zero-knowledge proof that all of the computations she sets up are properly constructed. If she attempts to cheat in any way which would cause Bob's final answer to deviate from a correct chaining, then with high probability, Bob will find out before the conclusion of the

**Protocol Chain**( $C_1, \dots, C_m, k$ )

- 1: Adding dummy inputs and outputs if necessary, ensure that for some  $n > k$ ,  $C_1, \dots, C_m$  give and receive  $n$  bit quantities to and from Bob. Alice and Bob agree on a  $2n + 1$  bit prime,  $p$ . This implicitly defines circuits

$$C_1[p]\{k\}, \dots, C_m[p]\{k\}.$$

For convenience, we denote  $C_i[p]$  by  $C_i^*$ . Using modulus  $p$ , Alice uniformly chooses keys  $K_1, \dots, K_{m-1}$ , and defines  $K_0, K_m = (1, 0)$ .

- 2: Alice and Bob successively evaluate circuits  $C_1^*, \dots, C_m^*$ , with security parameter  $k$ . Bob chooses his checksum inputs according to the rules given in Section 5.4. His normal input to  $C_1^*$  is equal to what he would normally have input to  $C_1$ , encoded as an element of  $Z_p$ . His normal input to  $C_i^*$  will be the output of  $C_{i-1}^*$ . Alice's input to  $C_i^*$  consist of the inputs she would have given to  $C_i$ , along with keys  $K_{i-1}^{-1}, K_i$ , which correspond to  $K_{in}^{-1}, K_{out}$  respectively. Each bit of these inputs are expanded into uniformly distributed  $k$ -expansions for that bit.
- 3: Alice and Bob run a checking circuit, *CHECK*, which we describe in Figure 5.10. If Bob sees a discrepancy, he concludes that the final output of circuit  $C_m^*$  is meaningless, otherwise he recovers the value that  $C_m$  would have output, up to trivial decoding from  $Z_p$  to bit strings.

Figure 5.9: A protocol for chaining together  $NC^1$  circuits.

**Circuit Check:** Circuit CHECK takes as input from Alice the  $k$ -expansions of all of Alice's keys,

$$K_{in}^{-1}[1], K_{out}[1], \dots, K_{in}^{-1}[m], K_{out}[m],$$

In protocol CHAIN, Alice makes the inputs  $K_{in}^{-1}[1], K_{out}[i]$  equal to the values of  $K_{in}^{-1}, K_{out}$  she input to circuit  $i$ . For each  $k$ -expansion input by Alice, CHECK takes a set of checksum queries from Bob, as per the consistency checking scheme of Section 5.4, and outputs the answers to these queries. In protocol CHAIN, Bob makes the same checksum queries for each  $k$ -expansion as he used before (As specified by the consistency checking scheme). Circuit CHECK also outputs a 1 iff the following three conditions hold.

1. Whenever  $K_{in}^{-1}[i]$  or  $K_{out}[i]$  are of the form  $(a, b)$ , then  $a \in Z_p^*$  and  $b \in Z_p$ . That is, all keys are legal.
2.  $K_{in}^{-1}[0] = K_{out}[m] = (1, 0)$
3. Whenever  $K_{out}[i-1] = (a, b)$ , and  $K_{in}^{-1}[i] = (c, d)$ , then  $ac \equiv 1 \pmod{p}$  and  $cb + d \equiv 0 \pmod{p}$ . That is, the keys are inverse.

Figure 5.10: Consistency checking circuit for use in the CHAIN protocol.

protocol, and effectively decide that Alice has stopped playing. The third property holds because Alice's view of each of the steps of the protocol is independent of Bob's input variables. This follows from the fact that this independence holds for the  $NC^1$  circuit evaluation protocol.

To prove the fourth property of the our protocol, we explicitly construct the simulator, which appears in Figure 5.11. For ease of presentation, we suppress the simulation of the consistency checking scheme. That is, we suppress the description of how Bob's checksum queries are simulated. For a description of how this is accomplished, we refer the reader back to Section 5.4.

**Lemma 5.2** Let  $k$  be sufficiently large. Then the output of simulator  $\text{CHAIN}(C_1, \dots, C_m, k)$  will be indistinguishable from the output of protocol  $\text{CHAIN}(C_1, \dots, C_m, k)$ , regardless of Bob's strategy.

**Proof:** For  $k$  sufficiently large, all of the simulated  $NC^1$  circuit evaluations will be indistinguishable from the actual circuit evaluations, provided that the input/output (I/O) behavior of the simulated circuits is indistinguishable from the input/output behavior of the actual circuits. It therefore suffices to prove that the behavior of the simulated Alice is indistinguishable from the behavior of the actual Alice. In other words, we must show that the I/O behavior of the simulated  $NC^1$  circuits is indistinguishable from the I/O behavior of the real circuits.

By our previous simulation results, we have that with high probability, Bob gets no information, in the information theoretic sense, about Alice's inputs to  $C_1^*, \dots, C_m^*$ . In particular, Bob doesn't learn anything about the encryption and decryption functions input by Alice than can be inferred from the I/O behavior of the circuits he has run so far. We exploit this fact in analyzing the I/O behavior of the actual circuits, for any strategy used by Bob. We allow Bob to be arbitrarily powerful, but do not allow him to have supernatural knowledge of the encryption and decryption functions.

The output of circuit  $C_1^*$  will be uniformly distributed, due to the unbreakability property of our encryption functions. Likewise, the output of the simulated circuit  $C_1^*$  is uniformly distributed.

To analyze the actual versus simulated behaviors of the rest of the  $C_i^*$ 's, there are two cases we must consider. In the first, "honest" case, we assume that the input given to circuit  $C_i^*$  is equal to the output of circuit  $C_{i-1}^*$ . In

**Simulator Chain**( $C_1, \dots, C_m, n, k$ )

- 1: The simulator, playing the part of Alice, agrees with Bob on a  $2n + 1$  bit prime  $p$ . The simulator then runs the subsimulator for  $\text{NC}^1\text{-EVAL}$  successively for circuits  $C_1[p]\{k\}, \dots, C_m[p]\{k\}$ .
- 2: The subsimulator for the evaluation of  $C_i[p]\{k\}$  will send the simulator a set of checksum queries, which are simulated according to Section consistency, and some normal inputs. In particular, the simulator for the evaluation of  $C_1[p]\{k\}$  will give Bob's initial input, which we denote by  $x_1, \dots, x_l$ . The simulator sends these values to Alice, who sends back

$$Z = C_m(C_{m-1}(\dots(C_2(C_1(X, Y_1), Y_2) \dots), Y_{m-1}), Y_m).$$

- 3: The simulator must also answer the normal queries of the subsimulator for circuit  $C_i[p]\{k\}$ . This normal output is simulated according to the following rules: We denote by  $Q_i$  the normal queries made by the subsimulator for  $C_i[p]$ , and by  $A_i$  the answer given to the subsimulator for  $C_i[p]$ .
  - a: If, for some  $i > 1$ ,  $Q_i \neq A_{i-1}$ , then  $A_i, \dots, A_m$  will consist of all 0's.
  - b: Whenever rule a doesn't hold, then  $A_i$  will be chosen uniformly from  $Z_p^*$ , for  $i \in [1, m - 1]$ , and  $A_m$  will be equal to  $Z$ .
- 4: The simulator then simulates the CHECK circuit, giving 1 as the normal output, and simulating all the checksum queries in the standard manner.

Figure 5.11: Simulator for the CHAIN protocol.

the “dishonest” case, there is some  $j$  such that the input to  $C_j^*$  deviates from the honest case for the first time.

In the honest case, the output of circuits  $C_2^*, \dots, C_{m-1}^*$  will be uniformly distributed, and the output of  $C_m^*$  will be equal to the output of circuit  $C$  (given the initial inputs to  $C_1^*$  and Alice’s inputs). This behavior is the same as that of the simulated circuits.

Now, suppose that the input to  $C_j^*$  is different from the output of  $C_{j-1}^*$ . By the unforgeability property of our encryption and decryption schemes, the output of  $C_j^*$  will be all 0’s, with probability at least  $1 - 2^{-n}$ . This holds regardless of Bob’s strategy. Now, if Bob fails to receive a valid encryption from circuit  $C_j^*$ , he will have at most a  $2^{-n}$  chance of giving a valid encryption to  $C_{j+1}^*$ . We can continue this argument for circuits  $C_{j+2}^*, \dots, C_m^*$ . By an elementary probability argument, we can thus bound above the probability that circuits  $C_j^*, \dots, C_m^*$  will output anything but 0’s by  $m2^{-n}$ .

The simulated circuits act in the same way, with one slight difference. As soon as a “dishonest” input is given, all the circuit outputs will be 0’s, with probability 1. However, for  $n$  sufficiently large, the discrepancy between events happening with probability  $m2^{-n}$  versus probability 0 is statistically indistinguishable. The lemma follows. ■

## Chapter 6

# Interactive Proof Systems with Multiple Provers.

### 6.1 Introduction.

#### 6.1.1 A popular history of proof systems.

*What is a proof?* This simple question opens the door to much turmoil and chaos. All professional mathematicians and theoretical computer scientists have some notion of what constitutes a proof, but do their notions agree? In the practical sense, certainly not. Some people will write down three words and an equation, and call it a proof. Others will write down twenty pages of meticulously crafted exposition before they are satisfied. Still others will have computers work out the tiny details of thousands of special cases, more than can be computed, or even verified by a human being. Often, the more loquacious individuals have little tolerance for the tersely presented papers, and those who favor parsimony often have equally little desire for wading through detailed proofs.

Still, despite these differences in practice, one can hope that researchers have some common ideal of what constitutes a proof. To a great extent they do. People can write down a fairly simple and intuitive set of axioms and inferences rules, and devise a mechanical procedure for checking that a sequence of axioms and inferences actually proves a theorem. True, there may be quibbles about which axioms to allow, fights over the axiom of choice and the continuum hypothesis, but for most purposes people can usually agree on

their ideal notion of proof. After much work and suffering, mathematicians achieved a certain measure of agreement.

And then came the theoretical computer scientists. They took the approach that it was not always enough to worry simply about the correctness of proofs. In order for a proof to be useful, it should be small enough to read and verify in a reasonable amount of time. Whereas in mathematics, rigor is in, and parsimony is out, theoretical computer science has the notion of an efficient proof, which requires both rigor and conciseness. Roughly, they require that an efficient proof of an assertion be of size polynomial in the size of the statement of the assertion.

From this revamped notion of proof springs forth the notion of NP, NP-completeness, and much of modern complexity theory. While still a curiosity to most mathematicians, there is a general consensus in the theory community that this is the “correct” notion of what constitutes an efficient proof.

Then came the theoretical cryptographers. They decided to extend the notion of efficient proofs. Ordinary proofs, they argue, are only one of many ways of convincing someone of the proof of an assertion. Instead of considering proofs as some static entity that can be written down on paper, cryptographers have developed the notion of proof systems in which an infinitely powerful prover interacts with a polynomially bounded verifier according to some prespecified rules. Furthermore, they relax the natural requirement that it should be impossible to convince someone of an incorrect assertion. Instead, they merely require that a charlatan has a nonnegligible probability of being caught.

It remains to be seen whether the cryptographic notion of proof will supplant the ordinary notion of proof. However, it has proven of inestimable use within this field. Furthermore, it is the first reasonable notion of proof which allows one to make a conceptually revolutionary distinction between the transfer of confidence and the transfer of knowledge. It is now both theoretically and practically possible to convince someone of certain assertions without imparting to that person any understanding about why the assertions is true. This is a paradoxical development, since the original notions of proof is a formalism for stating precisely why a theorem is true.

The goal of this chapter is to further generalize the notion cryptographic notion of proof systems. Instead of having a single infinitely powerful prover talking to a verifier, we consider scenarios in which we have several infinitely powerful provers. We will consider the power of such models, and show

that suffices to work with much more restricted versions of our model. Most importantly, we will analyze the knowledge complexity of our proof systems. We will show that in our model there is a complete split between confidence transfer and knowledge transfer. Any assertion that can be proved in our model can be proved in a manner which releases no knowledge to the verifier. This is the first such model for which this property has been rigorously proved.

### **6.1.2 Outline of this chapter.**

In Section 6.2 we formally and informally define our model. In Section 6.3, we show that anything provable by  $k$  provers is provable by only 2 provers. In Section 6.4, we prove a one-sideness result for 2-prover protocols, analogous to the one-sideness results for single prover interactive proof systems. In Section 6.5, we show that anything provable can be proved in statistical zero-knowledge, modulo an assumption about the existence of commitment with zero-knowledge proof schemes. In Section 6.6 we show how to implement oblivious transfer, and thus establish our zero-knowledge result without any assumptions.

## **6.2 The $k$ -Prover Model: Formal and Informal Definitions.**

There are two ways of expositing our new model. We can give formal definitions, or we can describe our models intuitively. In this chapter, we will do both. First we give our formal definitions, which must be written down, but are not meant to be read, and then we will describe what these definitions are trying to capture.

### **6.2.1 A Review of interactive proof systems and the notion of zero-knowledge.**

For the sake of completeness, we review the classical definitions of interactive proof systems and zero-knowledge, first put forth by Goldwasser, Micali, and Rackoff[GMR].

### Interactive proof systems.

**Definition 6.1** [GMR] Let  $P$  be a Turing machine which is computationally unbounded and  $V$  be a probabilistic polynomial-time Turing machine. Both machines have a read-only input tape, a work tape, and a random tape.  $P$  has a write-only communication tape on which it writes messages for  $V$ .  $V$  has a write-only communication tape on which it writes messages for  $P$ . The input, which we will typically denote by  $x$ , is written on the input tape of both the provers and the verifier. We call the ordered pair  $(P, V)$  an *interactive protocol*.

In other words, we consider a model in which a probabilistic polynomially bounded verifier interacts with an infinitely powerful prover. Like all Turing machines,  $V$  has a special *halt* or *accept* state. Thus, we have some notion of  $V$ , based on its interaction with  $P$ , accepting or not accepting a string  $x$ . This allows us to define what it means for a language  $L$  to have an interactive proof system.

**Definition 6.2** Let  $L \subseteq \{0, 1\}^*$ . We say that  $L$  has an *interactive proof system* (IPS) if there exists an interactive BPP machine  $V$  such that:

1.  $\exists P$  such that  $(P, V)$  is an interactive protocol, and  $\forall x \in L$ ,

$$\text{prob}(V \text{ accepts input } x \text{ after interacting with } P) \geq \frac{2}{3}$$

2.  $\forall x \notin L$  and  $\forall \hat{P}$  such that  $(\hat{P}, V)$  is an interactive protocol,

$$\text{prob}(V \text{ accepts input } x \text{ after interacting with } \hat{P}) \leq \frac{1}{3}$$

In other words, whenever  $x \in L$ ,  $P$  is able to convince  $V$  of this fact with probability at least  $\frac{2}{3}$ , and whenever  $x \notin L$ ,  $P$  is only able to deceive  $V$  with probability at most  $\frac{1}{3}$ .

### Zero-knowledge protocols.

Another key concept introduced in [GMR] is that of zero-knowledge protocols. Intuitively, an interactive protocol is zero-knowledge if no verifier can

learn anything from the protocol that it could not have learned before engaging in the protocol. We in fact require something even stronger: It must be possible to completely simulate any verifier's conversation with the prover. Another, more technical requirement, is that we allow malicious verifier's to have some arbitrary polynomial-sized advice tape to help them in their conversations. The formal definition is as follows.

**Definition 6.3** Let  $(P, V)$  be an interactive proof-system for  $L$ . Let  $View_{P,V(h)}(x)$  denote the verifier's view during the protocol (i.e. the verifier's coin tosses and the sequence of messages exchanged between the verifier and the prover), with auxiliary input  $h$ . This is a probability space taken over the coin tosses of  $V$ . We say that  $(P, V)$  is a *statistical zero-knowledge* protocol if for all probabilistic, polynomially bounded (PPT) verifiers  $\hat{V}$ , there exists a probabilistic Turing machine  $M$  which simulates  $\hat{V}$ . That is, for all  $x \in L$ , and for all advice strings  $h$ , of size at most  $|x|^\alpha$ , where  $\alpha$  is some fixed constant, the distribution  $M(x, h)$  (taken over  $M$ 's random tape) is statistically indistinguishable from  $View_{P,\hat{V}(h)}(x)$ . We also require that  $M(x, h)$  terminate in expected polynomial time.

The advice string,  $h$ , is added to the definition for very technical reasons. Basically, it allows one to prove closure properties for zero-knowledge protocols which would otherwise be false or difficult to prove. These technical considerations don't come up in our model, and so we will delete any mention of an auxiliary input. We could, if we wished, stick the advice functions into our model without altering any of our proofs.

## 6.2.2 Formal definitions of the multi-prover model.

**Definition 6.4** Let  $P_1, \dots, P_k$  be Turing machine which are computationally unbounded and  $V$  be a probabilistic polynomial-time Turing machine. All machines have a read-only input tape, a work tape, and a random tape. In addition,  $P_1, \dots, P_k$  share a read-only random tape in common. Each  $P_i$  has a communication tape on which it writes messages for  $V$ .  $V$  has  $k$  communication tapes on which it writes messages for the provers. On communication tape  $i$ ,  $V$  writes messages to  $P_i$ . We call the  $(k + 1)$ -tuple  $(P_1, \dots, P_k, V)$  a *k-prover interactive protocol*.

**Definition 6.5** Given an input  $x$  and interactive proof system

$$(P_1, \dots, P_k, V),$$

we define  $\text{accept}(P_1, \dots, P_k, V, x)$  as the probability that  $V$  accepts input  $x$  after interacting with  $P_1, \dots, P_k$ .

**Definition 6.6** Let  $L \subseteq \{0, 1\}^*$ . We say that  $L$  has a *k-prover interactive proof system* (*k-IPS*) if there exists an interactive BPP machine  $V$  such that:

1.  $\exists P_1, \dots, P_k$  such that  $(P_1, \dots, P_k, V)$  is an interactive protocol, and  $\forall x \in L$ ,

$$\text{accept}(P_1, \dots, P_k, V, x) \geq \frac{2}{3}$$

2.  $\forall x \notin L$  and  $\forall \hat{P}_1, \dots, \hat{P}_k$  such that  $(\hat{P}_1, \dots, \hat{P}_k, V)$  is an interactive protocol,

$$\text{accept}(\hat{P}_1, \dots, \hat{P}_k, V, x) \leq \frac{1}{3}$$

**Definition 6.7** We denote by  $IP_k$  the class of languages which have *k-prover interactive protocols*.

We should note here that the  $\frac{2}{3}, \frac{1}{3}$  split between the acceptance probabilities for strings in and out of the language is purely arbitrary. We could have just as well have required a split of  $\frac{1}{2} + \epsilon, \frac{1}{2} - \epsilon$  or a split of  $1 - \epsilon, \epsilon$ . Fortunately, by the same argument as with ordinary interactive proof systems, these restrictions are all equivalent. Given a *k* prover interactive proof system which achieves a  $\frac{1}{2} + \epsilon, \frac{1}{2} - \epsilon$  split, we can convert it to a protocol which achieves a  $1 - \epsilon, \epsilon$  split. We accomplish this by have the verifier run the protocol polynomially many times, *in series*, and accept if and only if he accepted for the majority of runs.

**Definition 6.8** Let  $(P_1, \dots, P_k, V)$  be a *k-prover interactive proof-system* for  $L$ . Let  $\text{View}_{P_1, \dots, P_k, V}(X)$  denote the verifier's view during the protocol (namely the verifier's coin tosses and the sequence of messages exchanged between the verifier and the provers). This is a probability space taken over the coin tosses of  $V$ . We say that  $(P_1, \dots, P_k, V)$  is a *statistical zero-knowledge*

protocol if for all probabilistic, polynomially bounded (PPT) verifiers  $\hat{V}$ , there exists a probabilistic Turing machine  $M$  such that for all  $x \in L$ , the distribution  $M(x)$  (taken over  $M$ 's random tape) is statistically indistinguishable from  $\text{View}_{P_1, \dots, P_k, \hat{V}}(x)$ , and  $M(x)$  terminates in expected polynomial time. We say that  $(P_1, \dots, P_k, V)$  is a *perfect zero-knowledge* protocol if  $M(x)$  realizes the exact distribution of  $\text{View}_{P_1, \dots, P_k, \hat{V}}(x)$ , and  $M(x)$  terminates in expected polynomial time.

### 6.2.3 Informal Definitions.

The best way of imagining  $k$ -prover interactive proof systems is to imagine  $k$  provers  $P_1, \dots, P_k$ , who are trying to convince a verifier of some assertion about some input,  $x$ . The verifier will toss a set of coins, and start posing questions to the provers. Based on the input  $x$  and his random coin-tosses, he poses a question to prover  $P_1$ . The prover will then give him some answer. Based on what he's seen (at this point his coin-tosses, the input, and the answer to his first question), he formulates a question for the second prover,  $P_2$ . Likewise, he proceeds to question  $P_3, \dots, P_k$ . He may then repeat this questioning process, to his hearts desire. Of course, his total running time must be polynomial in the size of an input.

Now, if the provers were allowed to confer with each other during the protocol, there would be no point in having more than one of them. Therefore, we impose the constraint that the provers are not allowed to confer with each other during the protocol. We do allow them to collude as much as they want before the beginning of the protocol. To do this, we allow them all to have a common set of coin tosses and the input  $x$ . It is a homework exercise to verify that this allows for essentially arbitrary colluding, since the provers can simulate all the pre-protocol actions of the other provers.

Our definition of when a language has a  $k$ -prover interactive proof system is just the natural extension of the ordinary definition, as a textual comparison will quickly reveal. Likewise, our definition for statistical zero-knowledge is just the ordinary one altered to accommodate our new notion of proof systems, and eliminating the advice strings in the definition.

### What about other definitions of zero-knowledge?

There are almost as many flavors of zero-knowledge as flavors of ice cream at Toscanini's.<sup>1</sup> One might wonder why we only attempted to extend the notion of statistical and perfect zero-knowledge to the multi-prover scenario. This is for an aesthetic reason: statistical and perfect zero-knowledge are the strongest definition of zero-knowledge we have. Computational zero-knowledge turns out to be a weaker and more difficult notion to work with. We use it in the single-prover model in favor of statistical or perfect zero-knowledge because we can then prove zero-knowledge properties for a much wider range of protocols. However, in the multi-prover model, we can make all our protocols achieve at least statistical zero-knowledge, so no weaker definition need be considered.

## 6.3 On the Power of Two Verses Many Provers.

In the last section, we introduced the  $k$ -prover model for interactive proof systems. It essentially replaced the single prover found in the ordinary model with many noncommunicating provers. The natural question to ask after creating this or any model is “What for?”. An interesting model should capture our intuition about some phenomena that is of interest to us, and should have some nontrivial properties that shed light on this phenomena. This is an admittedly (perhaps purposely) vague statement which we will nevertheless try to adhere to in the coming sections.

### 6.3.1 Why should many provers more powerful than one?

It is not immediately obvious why having more than one prover should alter in any way the class of assertions which can be proven. Intuitively, one might argue, one infinitely powerful prover should be enough. Without proving anything one way or another, we give some intuition for why having more than one prover should help.

---

<sup>1</sup>Toscanini's is an ice cream parlor at Cambridge. This is *not* a compensated endorsement.

It is common for police to interrogate suspected criminals.<sup>2</sup> On good days, they will have two suspects instead of just one, in which case they will invariably separate them during questioning. The reason for this is to catch inconsistencies. If the two suspects were together, they could easily corroborate each other's testimony. However, if suspect  $A$  was not aware of suspect  $B$ 's answer to a question, he would not necessarily be able to back him up.

Now, if it is harder to convince someone of a falsehood without being detected, this will enhance one's credibility if one isn't caught. Intuitively, this heightened credibility may allow one to convince someone of harder to believe assertions. If lie detectors really existed, then passing such a test would lend credibility to whatever one had said. Likewise, if questioning two suspects apart from each other is more likely to reveal inconsistencies in their story, then this procedure will also lend greater credibility if they do give consistent answers. A story that is not believable when told by a single person may be more believable when two people give independent, corroborating reports. The intention of this informal argument is merely to suggest the source of whatever extra power we may get. Unfortunately, whether many provers can prove more than two is still very open.

Most of our belief that they can indeed prove more comes from what we can't prove. For instance, we can't prove any analog to the result of Goldwasser-Sipser[GS], which says that  $IP = AM^3$ . This implies that all interactive proofs can be reduced to probabilistic games of complete information. There is no such equivalence known for the multi-prover case, and so these proof systems seem to possess more of the flavor of games of incomplete information. Games of incomplete information seem to be more complex than games of complete information, and this complexity may be exploitable in terms of language recognition power.

Furthermore, we cannot even prove that  $IP_2$  is contained in PSPACE. This is true even if the verifier simply sends random bits to each prover, and then computes some polynomial-time predicate based on the input  $x$ , the random strings sent, and the answers received. In the single-prover model,

---

<sup>2</sup>Likewise for civil rights activists, anti-nuke demonstrators, nonwhites, etc., but since I would never dream of injecting politics into my thesis, I won't go into this.

<sup>3</sup>The complexity class  $AM$ , proposed by Babai[Bab], is like interactive proofs, except that in  $AM$  protocols, all the questions given by the verifier are randomly generated by a fair coin.

however, it is not hard to prove that  $IP \in \text{PSPACE}[F]$ .

### 6.3.2 Two provers are as many as we need.

Given our belief that two provers can prove more than one, it seems reasonable to also believe that three provers can prove more than two. Likewise, we might believe that for all  $i < j$ ,  $IP_i$  will be properly contained in  $IP_j$ . However, it turns out that  $IP_k = IP_2$  for  $k \geq 2$ .

To prove this theorem, the following definitions are useful.

**Definition 6.9** We define a history  $H$  to be finite sequence of ordered pairs of strings. We denote by  $\mathcal{H}$  the set of histories. Thus, if  $H \in \mathcal{H}$ , then  $H$  is of the form  $(q_1, a_1), (q_2, a_2), \dots, (q_i, a_i)$ .

**Definition 6.10** For a fixed input,  $x$ , we define a *strategy*  $S$  to be a function from  $\mathcal{H} \times \Sigma^*$  to distributions on  $\Sigma^*$ . In other words, a strategy  $S$  takes as input  $[(q_1, a_1), \dots, (q_{i-1}, a_{i-1})], q_i$  and produces  $a_i$  according to some distribution. A strategy is deterministic if its range consists of distributions on single elements. That is, given an input string, a partial conversation, and a question, it will always produce the same answer.

**Remark:** If strategy  $S$  is deterministic, the values of  $a_1, \dots, a_{i-1}$  may be reconstructed from  $q_1, \dots, q_{i-1}$ . Thus, for deterministic strategies  $S$ , we may equivalently represent it as a function that takes a query sequence  $q_1, \dots, q_i$ , and outputs answer  $q_i$ .

The above definitions are merely a formalization of what we mean by a prover's strategy. We will freely identify a prover with its strategy.

The following easy to prove lemma simplifies our analysis. It states that in many circumstances we can assume that all the provers' strategies are deterministic.

**Lemma 6.1** Let  $(P_1, \dots, P_k, V)$  be an interactive protocol such that on input  $x$  the verifier accepts  $x$  with probability  $p_x$ . Then there exist  $\hat{P}_1, \dots, \hat{P}_k$  such that

1.  $\hat{P}_1, \dots, \hat{P}_k$  are deterministic.
2. On input  $x$ ,  $(\hat{P}_1, \dots, \hat{P}_k, V)$  will accept with probability at least  $p_x$ .

**Proof:** The proof of this lemma proceeds in two steps. First, we convert a protocol in which the provers use arbitrary probabilistic strategies to one in which they, at the beginning of the protocol, randomly choose a deterministic strategy which they abide by for the duration of the protocol. This transformation will leave unchanged the probability that the verifier accepts  $x$ . We then transform this protocol into a purely deterministic one (from the provers' point of view).

To accomplish the first transformation, we can just have each prover  $P_i$  exhaustively go through every possible history  $\mathcal{H}$  and question  $q$ , and sample its response,  $a$ , according to strategy  $S_i$ . Note that there are only a finite set of histories and questions, since the verifier has some polynomial bound on the number of steps it may execute. Prover  $P_i$  can then use these sampled responses for the rest of the protocol. Clearly, it cannot matter to the verifier whether the random choices made by  $P_i$  were made on the spot or decided on beforehand. Thus, the probability that the verifier accepts  $x$  will be unchanged.

After the first transformation, we have the following scenario. On input  $x$ , deterministic strategies  $S'_1, \dots, S'_k$  are chosen according to some distribution. The expected value of  $\text{accept}(S'_1, \dots, S'_k, V, x)$  will be  $p$ . Now, by a standard probability argument, there must be fixed deterministic strategies  $S''_1, \dots, S''_k$  such that  $\text{accept}(S''_1, \dots, S''_k, V, x)$  is no less than the expected value, that is,

$$\text{accept}(S''_1, \dots, S''_k, V, x) \geq p.$$

The lemma follows. ■

Lemma 6.1 will simplify our collapsing theorem, because now we can assume that all the provers, good or bad, are using deterministic strategies. This will allow a single prover to state precisely what another prover will do under a given set of circumstances.

We cannot so simplify the verifier's strategy. If the verifier's strategy is deterministic, then interaction can't increase the set of recognizable languages beyond NP. However, we can use the standard trick of viewing the verifier as first sampling its set of random bits, and then acting deterministically based on these bits. This corresponds to the first step of the above reduction for the provers.

We now prove the main theorem of this section.

**Theorem 6.1**  $IP_k = IP_2$ .

**Historical Note:** The above theorem was first proved in [BGKW]. Subsequently, independently, this theorem was also proved by Fortnow, Rompel, and Sipser[FRS].

**Proof:** Given an interactive protocol  $(P_1, \dots, P_k, V)$  for a language  $L$ , we produce a two-prover protocol  $(P'_1, P'_2, V')$  for  $L$ . We accomplish this transformation in two stages. First, we create a 2-prover protocol that partially achieves the requirements of interactive proof systems. We then show that by running this protocol many times in series we can create a 2-prover interactive proof system for  $L$ .

On input  $x$ , where  $|x| = n$ , let  $S_i$  be the strategy used by  $P_i$ , which without loss of generality we assume is deterministic. Likewise, we also assume without loss of generality that  $V$  asks exactly  $m$  questions of each prover, where  $m$  is polynomial in  $n$ .

The core of our reduction is given in Figure 6.1. At the end of running this 2-prover protocol, the verifier  $V$  outputs either “good,” “bad,” or “ugly.” Intuitively,  $V'$  outputs “good” if it detects no cheating, and determines that  $V$  would have accepted given the simulated conversation. Similarly,  $V'$  outputs “bad” if it detects no cheating, but determines that  $V$  would have rejected. It outputs “ugly” if it detects cheating by the provers. A judgment of “good” is positive evidence, though not conclusive, that  $x \in L$ , and a judgment of “bad” is negative evidence, though not conclusive, that  $x \in L$ . A judgment of “ugly,” however, is conclusive evidence that the provers are cheating.

Now if  $x \in L$ , then the provers will act as specified, and thus  $(P'_1, P'_2, V')$  will faithfully simulate the original protocol. Thus,  $V'$  will output “good” with probability at least  $\frac{2}{3}$ , “bad” with probability  $\frac{1}{3}$ , and “ugly” with probability 0.

If  $x \notin L$ , then we would ideally wish to show that, for any provers  $\hat{P}_1, \hat{P}_2$ , verifier  $V'$  would output “good” with probability at most  $\frac{1}{3}$ . This would prove our theorem immediately. Unfortunately, we cannot make this guarantee. A malicious  $\hat{P}_1$  can use its knowledge of the simulated  $V$ ’s random coin tosses,  $R$ , to increase its chance of getting  $V$  to accept. What we can prove is that if  $V'$  outputs “good” too often, then it will also output “ugly” with nonnegligible probability.

**Lemma 6.2** Let  $\hat{P}_1, \hat{P}_2$  be deterministic provers, such that  $(\hat{P}_1, \hat{P}_2, V)$  is an interactive protocol. Let  $x \notin L$ . Then with probability  $\frac{2}{3}$ , bits  $R$  will be such that one of the two cases must hold.

**Protocol 2-Prover-Partial( $P_1, \dots, P_k, V, x$ )**

- 1:** Verifier  $V'$  uniformly chooses a string  $R$  of coin tosses that would have been used by verifier  $V$ . Thus,  $R$  corresponds to the contents of  $V$ 's random tape. It sends  $R$  to  $P'_1$ .
- 2:** Prover  $P'_1$  computes the entire transcript of the conversation  $V$  would have had with  $P_1, \dots, P_k$ . This is possible since the provers' strategies are deterministic, as is  $V$ 's strategy once  $R$  has been selected. Let  $q_j^i$  be the  $j$ th question  $V$  asks  $P_i$ , and let  $a_j^i$  be the  $j$ th answer  $V$  receives from  $P_i$ . Prover  $P'_1$  sends this transcript to  $V'$ .
- 3:** Verifier  $V'$  uniformly selects  $i \in [1, k]$ , and  $j \in [1, m]$ . It sends  $i, q_1^i, \dots, q_j^i$  to  $P'_2$ .
- 4:** Prover  $P'_2$  sends  $a_j^i$  to  $V'$ .
- 5:** Verifier  $V'$  outputs either "good," "bad," or "ugly," according to the following rules.
  - If the transcript given to  $V'$  has  $V$  asking a question different from what it would ask given the conversation so far and  $V$ 's coin tosses, then  $V'$  outputs "ugly."
  - If  $P'_2$  gives anything but  $a_j^i$  in Step 4,  $V'$  outputs "ugly."
  - Otherwise,  $V'$  outputs "good" if  $V$  would have accepted given the transcript and its coin tosses, and "bad" if  $V$  would have rejected.

Figure 6.1: Core protocol for simulating  $k$  provers with 2.

1. Prover  $\hat{P}_1$  acts in such a way that verifier  $V'$  will output “bad” with probability 1.
2. Prover  $\hat{P}_1$  acts in such a way that verifier  $V'$  will output “ugly” with probability at least  $1/mk$ . Recall that  $k$  is the number of provers, and  $m$  is the number of questions asked of each prover.

**Proof:** First, we give the intuition behind our protocol. The good prover,  $P'_1$ , outputs a transcript of the conversations that  $P_1$  would have had with verifier  $V$  if the verifier’s random coin tosses were equal to  $R$ . We cannot say so much about the transcript output by a bad prover,  $\hat{P}_1$ . Indeed, the transcripts produced by  $\hat{P}_1$  may not be consistent with the input/output behavior of any real set of provers, since it may use knowledge unavailable to a real prover. However, prover  $\hat{P}_2$  is not allowed such foresight, and thus its input/output behavior does implicitly define a set of provers  $P_1^*, \dots, P_k^*$ . Intuitively, prover  $\hat{P}_1$  must make his transcripts consistent with  $P_1^*, \dots, P_k^*$ , or be caught with nonnegligible probability. this motivates the following definitions.

**Definition 6.11** Fix input  $x$ , and let  $P_1, \dots, P_k$  be deterministic provers, such that  $P_i$  uses strategy  $S_i$  on input  $x$ . Let  $V$  be a probabilistic verifier, and let  $R$  be a sequence of coin tosses. We define  $\text{transcript}(S_1, \dots, S_k, V, R, x)$  to be the transcript of the conversation that would ensue if protocol  $(P_1, \dots, P_k, V)$  were run on input  $x$ , and the verifier’s sequence of coin tosses were equal to  $R$ .

**Definition 6.12** Let  $V$  be a probabilistic verifier, and let  $R$  be a sequence of coin tosses. We say a transcript  $\tau$  is  $(V, R)$ –consistent if all the questions asked by the verifier in  $\tau$  are equal to what  $V$  would have asked given that,

1. Verifier  $V$ ’s sequence of coin tosses is equal to  $R$ .
2. The earlier portion of the conversation, as given by  $\tau$ , had actually happened.

By definition,  $\text{transcript}(S_1, \dots, S_k, V, R, x)$  is  $(V, R)$ –consistent.

Note that if  $V$  runs in probabilistic polynomial time, then testing if a transcript  $\tau$  is  $(V, R)$ –consistent is in  $P$ . This test is embodied in the first

condition of Step 5 in protocol 2-PROVER-PARTIAL. It is also not hard to see that if  $\tau_1, \tau_2$  are both  $(V, R)$ -consistent, but  $t_1 \neq t_2$ , then they must have one of the provers give different answers given the same sequence of previous questions to it.

We now resume with the main part of the proof. With input string  $x$ , let strategy  $S_i^*$  be defined as follows. On input  $((q_1, a_1), \dots, (q_i, a_i)), q_{i+1}$ ,  $S_i^*$  outputs the answer given by  $\hat{P}_2$  when asked  $i, q_1^i, \dots, q_j^i$  on input  $x$ .

On input  $x \notin L$ , and with bit sequence  $R$ , prover  $\hat{P}_1$  can output  $\text{transcript}(S_1^*, \dots, S_k^*, V, R, x)$ , or something else. By the definition of  $k$ -prover interactive proof systems, we know that for any set of prover strategies  $S_1^*, \dots, S_k^*$ , only  $\frac{1}{3}$  of the  $R$ 's will cause  $\text{transcript}(S_1^*, \dots, S_k^*, V, R, x)$  to have the verifier accept. Thus, for  $\frac{2}{3}$  of  $R$ 's, the only way in which the verifier can possibly output "good" is for  $\hat{P}_1$  to give some transcript  $\tau$  which is not equal to  $\text{transcript}(S_1^*, \dots, S_k^*, V, R, x)$ . It suffices then to show that whenever this happens,  $V'$  will output "ugly" with probability  $\geq 1/mk$ .

For ease of exposition, let  $q_j^i$  be the  $j$ th question asked by the verifier to prover  $i$ , according to  $\text{transcript}(S_1^*, \dots, S_k^*, V, R, x)$ , and let  $a_j^i$  denote the  $i$ th prover's answer to this question. Similarly, let  $q_j^i[\tau]$  be the  $j$ th question asked by the verifier to prover  $i$ , according to  $\tau$ , and let  $a_j^i[\tau]$  denote the corresponding answer.

We can assume without loss of generality that  $\tau$  is  $(V, R)$ -consistent, since otherwise, the verifier would output "bad." Hence, there must be some  $i \in [1, k]$  and  $j \in [1, m]$  such that  $q_l^i = q_l^i[\tau]$  for  $l \in [1, j]$ , but  $a_j^i \neq a_j^i[\tau]$ . However, with probability  $1/mk$  the verifier will send  $\hat{P}_2$  the values  $i, q_1^i, \dots, q_j^i$ . Then, by definition of  $S_i^*$ , the answer given by  $\hat{P}_2$  will be equal to  $a_j^i$ , and the verifier will output "bad." Hence,  $V'$  will output "bad" with probability at least  $1/mk$ . The lemma follows. ■

Using Lemma 6.2, we now exhibit the complete transformation from a  $k$ -prover to a 2-prover protocol, given by the protocol in Figure 6.2, and prove its correctness.

Now, if  $x \in L$ , then each iteration of protocol 2-PROVER-PARTIAL will output "good" with probability at least  $\frac{2}{3}$ . Thus, with very high probability, the verifier will output "good" at least  $4kmn$  times, as  $n$  grows large.<sup>4</sup>

---

<sup>4</sup>Note that we do not care about values of  $x$  whose length is not sufficiently large. This possibly troublesome set is finite, and thus easily dealt with.

**Protocol 2-Prover-Complete( $P_1, \dots, P_k, V, x$ )**

1:  $P'_1, P'_2, V'$  execute protocol 2-PROVER-PARTIAL( $P_1, \dots, P_k, V, x$ )  $8kmn$  times in series. If  $V'$  ever outputs “ugly,” it aborts the protocol and rejects. Otherwise, it accepts  $x$  iff it outputs “good” more than  $4kmn$  times.

Figure 6.2: Driver protocol for simulating  $k$  provers with 2.

If, on the other hand,  $x \notin L$ , we wish to argue that with high probability, either not enough iterations of the protocol will output “good,” or one of the iterations will output “ugly.” With high probability, only  $3kmn$  of the iterations of the protocol will allow the first prover to output a faithful simulation and still have the verifier accept. Thus, with high probability, the first prover would have to run  $kmn$  cheating simulations in order to make the verifier accept  $x$ . In this case, the verifier will output “ugly” for one of these simulations with probability

$$1 - (1 - mk)^{mn} \geq 1 - 2^{-\Theta(n)},$$

for  $n$  large. Thus, with high probability, the verifier will reject  $x$ , and we are done. ■

### 6.3.3 Discussion, and open problems.

#### Can we collapse rounds as well as provers?

The theorem we have just proved illustrates two nontrivial facts about language recognition in the multi-prover model. First and foremost, it is interesting, though in hindsight not unsurprising, that having more than two provers doesn't help. Philosophically, the ability to check provers against each other, while keeping them all somewhat in the dark, has a qualitative effect. Once the proof system model is complicated enough to have this property at all, it reaps the full advantage there is to be had by it.

Secondly, the subprotocol 2-PROVER-PARTIAL effects a collapse not just in the number of provers, but in the number of rounds of communication as well. The essence of a proof system, if not the technically correct acceptance/rejection probabilities, is captured by this protocol. The amplification

obtained by repeatedly running the protocol is a purely mundane step; the magic has already occurred. Indeed, if one could obtain such properties in a constant number of rounds in the standard model, one could collapse  $IP$  to  $IP[2]$  (interactive proofs with only two rounds of communication) by running the protocol many times in parallel. However, as is shown in [AGH], this is almost certainly not true.

One might ask, in fact, why one couldn't simply run 2-PROVER-PARTIAL many times in parallel, thus giving constant round interactive proofs. Standard interactive proofs can be run in parallel with roughly the same amplification in security as when they are run serially. However, this amplification property breaks down in the case of multiple provers. One of the more interesting unsolved problems in this area is whether there is a cleverer scheme for amplifying the acceptance/rejection probabilities of 2-prover proof systems without increasing the number of rounds of communication.

### **The notion of robustness for protocol transformations.**

In this section, we showed how to perform two forms of protocol transformations. We showed (existentially) how to convert a proof system with probabilistic provers into a proof system with deterministic provers, and we showed how to convert a multi-prover protocol into a 2-prover protocol. In all these transformations, the verifier remains a probabilistic polynomial time machine, and is thus not required to grow significantly more powerful. One may ask whether the transformation leaves the power of the provers essentially unchanged as well. Roughly, we call a transformation which leaves the power of the provers unchanged a *robust* protocol.

Just what it means to leave the power of the provers unchanged is open to interpretation. One tack would be to say that each new prover  $P'_i$  must consist of a probabilistic polynomial machine which is allowed to, in the course of the execution of the protocol, make polynomially many queries of each of the original provers. In other words, a new prover,  $P'_5$ , might make a sequence of queries to the old prover  $P_7$ , restart it and make a new sequence of queries, back it up one step and make a different last query, then make a sequence of queries to the old  $P_2$ , etc. In this case, we say the protocol is *weakly robust*.

A stronger requirement would be to only allow a new prover to have polynomial accesses to the prover it is replacing (this notion is only applicable

if a  $k$ -prover protocol is being transformed into a  $k$ -prover protocol). Thus, the new prover  $P'_5$  can only use the old  $P_5$ . We call such a transformation *robust*.

The strongest notion along these lines is to require that each new prover,  $P'_i$ , is a probabilistic polynomial time machine which is only allowed to have a single conversation with the old  $P_i$ . We call such a transformation *strongly robust*.

Intuitively, robust transformations are much more realistic. In real protocol situations, the “provers” are not infinitely more powerful at all. Instead, they may have some secret information, or access to some fairly specific extra ability. The notion of infinitely powerful provers is interesting from the standpoint of complexity theory, and is useful as a way of modeling adversaries with unknown abilities. However, it should not be considered a realistic requirement to place on those who must implement the protocols. Protocols should be executable by relatively weak entities, and safe against infinitely powerful entities.

Having made a set of requirements, it is interesting to see if we can actually fulfill them. Unfortunately, this is hard. Our transformation from probabilistic provers to deterministic ones does not fit any of these notions of robustness. Each prover is essentially required to compute a deterministic strategy which is as good as the random one. While we have an existential proof that such a deterministic strategy exists, finding it could take a great deal of time.

All of the protocol transformations that will be described in this thesis implicitly use the probabilistic to deterministic transformation, and thus also fail to be robust. This weakness is disappointing, since it makes these results existential in nature. Thus, we prove that if there is a  $k$ -prover interactive proof system for  $L$ , then there is a 2 prover interactive proof system for  $L$ . However, being able to implement a  $k$ -prover interactive proof system for  $L$  does not allow one to implement a 2-prover interactive proof system for  $L$ .

This difficulty is not unique to the multi-prover scenario. Neither the original protocol transformation used by Goldwasser–Sipser to show that  $IP = AM$ , nor the simpler transformation due to Kilian, is robust. Because of this, neither the protocol transformation used by Implagliazzo–Yung, to show that  $IP$  is in zero-knowledge, nor a later transformation, discovered by Ben-Or–Goldreich–Goldwasser–Håstad–Kilian–Micali–Rogaway[BGGHKMR], are robust. This is a rather depressing state of affairs,

which has received remarkably little attention. The problem of making protocol transformations robust, or showing that this would be difficult, is still very open.

Interestingly, it is possible to robustly transform any 2-prover interactive proof system for  $L$  into a 2-prover interactive proof system which is in statistical zero-knowledge. Indeed this transformation is strongly robust, in the sense outlined above. Due to space concerns, this thesis contains only the much simpler transformation, which is not robust. The robust transformation, which requires the full use of our oblivious transfer machinery, coupled with some results about communication complexity, is briefly outlined in [BGKW].

## 6.4 A One-sidedness Result for 2-Prover Proof Systems.

### 6.4.1 Definitions, and statement of the main theorem.

In an interactive proof system for some language  $L$ , the verifier can make two types of error. A verifier can either

1. Accept a string  $x$  which is not in  $L$ , or
2. Reject a string  $x$  which is in  $L$

It is not hard to see that the first type of error can be eliminated exactly for those languages which are in  $NP$ . Hence, this case is not interesting. The second case is much more interesting, and thus warrants a definition.

**Definition 6.13** Let  $(P_1, \dots, P_k, V)$  be an interactive proof system for some language  $L$ . We say that  $(P_1, \dots, P_k, V)$  is *one-sided* if

$$(\forall x \in L) \text{accept}(P_1, \dots, P_k, V, x) = 1.$$

In short, an interactive proof system is one-sided if the verifier never mistakenly rejects a good string. For the case of a single prover, we have the classic theorem of Goldreich, Mansour, and Sipser:

**Theorem 6.2** [GMS] Let  $L$  be an arbitrary language in  $IP(= IP_1)$ . Then there exists a one-sided interactive proof system  $(P, V)$  for  $L$ .

Thus, any language  $L$  that has an ordinary (one prover) interactive proof system also has a one-sided interactive proof system. A natural question to ask then is whether this theorem holds for proof systems which have more than one prover. In this section, we answer this question in the affirmative. Specifically, we prove:

**Theorem 6.3** Let  $L$  be an arbitrary language in  $IP_2$ . Then there exists a one-sided interactive proof system  $(P_1, P_2, V)$  for  $L$ .

Note that we need not prove analogous theorems for  $IP_3, IP_4, \dots$ , since by the result of the previous section these language classes are equal to  $IP_2$ . **Proof:** Our proof is completely derivative of the techniques of [GMS]. First, we show how to reduce an assertion about language membership into an assertion about the size of a certain set. We then use a lemma due to Lautman to show how to prove these assertions in a one-sided fashion.

In our transformation from ordinary interactive proof systems to one-sided interactive proof systems, we need to prove assertions about the coin tosses used by the original verifier. To this end, we make the following definition:

**Definition 6.14** Let  $(P_1, \dots, P_k, V)$  be a multi-party protocol, where provers  $P_1, P_2$  are deterministic. Let  $b(n)$  be a polynomial upper bound on the number of random bits used by  $V$  on an input of size  $n$ . For any input  $x$ , we define  $accepting(P_1, \dots, P_k, V, x, b)$  to be the set of bit strings  $R$ , of length  $b(|x|)$ , such that  $V$  would accept  $x$  if its random tape consisted of  $R$ .

The following identity follows immediately from the above definition:

$$accept(P_1, \dots, P_k, V, x) = \frac{|accepting(P_1, \dots, P_k, V, x, b)|}{2^{b(|x|)}}.$$

Thus, given a proof system for an arbitrary language  $L$ , asserting that  $x \in L$  is equivalent to asserting that  $accepting(P_1, \dots, P_k, V, x, b)$  is large. The key idea in our transformation is to use a lemma due to Lautman to give a one-sided proof that a set is large.

### 6.4.2 Covering sets and Lautman's lemma.

Before stating Lautman's lemma, some new terminology is useful. Lautman's lemma applies to general sets, so the definitions below will not explicitly refer to our definitions above. For continuity, note that we will always have set  $S$ , be equal to  $\text{accepting}(P_1, \dots, P_k, V, x, b)$ .

**Definition 6.15** Let  $S, T$  be subsets of  $\{0, 1\}^n$ . Let  $r \in \{0, 1\}^n$ . We define the function  $|\text{hit}(S, T, r)|$  to be those  $t \in T$  such that  $t \oplus r \in S$ , where the  $\oplus$  operator performs componentwise exclusive-or.

**Definition 6.16** Let  $S, T$  be subsets of  $\{0, 1\}^n$ . We say that  $T$  is a *covering set* for  $S$  if, for all  $r \in \{0, 1\}^n$ ,  $|\text{hit}(S, T, r)| > |T|/2$ .

We are interested in classes of sets for which covering sets provably do or do not exist. A trivial observation to make is that if  $|S| \leq 2^{n-1}$ , no covering set  $T$  exists. If  $r$  is distributed uniformly over  $\{0, 1\}^n$ , the expected value of  $|\text{hit}(S, T, r)|$  will be equal to

$$\frac{|S| \cdot |T|}{2^n},$$

by the linearity of expectation. For  $|S| \leq 2^{n-1}$ , this is at most  $|T|/2$ . A useful corollary of the above identity is that

$$\text{prob} \left( |\text{hit}(S, T, r)| \geq c \cdot \frac{|S| \cdot |T|}{2^n} \right) \leq \frac{1}{c}.$$

If  $|S| > 2^{n-1}$ , and  $T = \{0, 1\}^n$ , then  $|\text{hit}(S, T, r)| > |T|/2$  for all  $r \in \{0, 1\}^n$ . Thus, the set of all strings in  $\{0, 1\}^n$  is a trivial covering sets. The following lemma of Lautman shows that every sufficiently large  $S$  has a small covering set.

**Lemma 6.3** [Lautman's Lemma] Let  $S$  be a subset of  $\{0, 1\}^n$ , and let  $|S| \geq \frac{2}{3} \cdot 2^n$ . Then, for some constant  $c$  independent of  $n$  and  $S$ , there exists a covering set  $T \subset \{0, 1\}^n$  of size at most  $n^c$ . This lemma holds when the factor of  $\frac{2}{3}$  is replaced by any fraction greater than  $\frac{1}{2}$ . ■

Lautman's lemma is useful to us in at least two ways. First, it converts a statement of the form "Property  $P$  holds for most values of  $x$ ," into a statement of the form "Property  $P'$  holds for *all* values of  $x$ ." It is this

conversion which will allow us to achieve one-sided protocols. Second, the lemma also allows us to differentiate large sets from small sets. If  $|S| \leq 2^n/3$ , then for all possible  $|T|$ ,

$$\text{prob}(|\text{hit}(S, T, r)| \geq |T|/2) \leq \frac{2}{3}.$$

Thus, if  $S$  is too small, no set  $T$  exists with the above two properties. Furthermore, any set  $T$  will be exposed with nonnegligible probability by a randomly chosen  $r$ .

### 6.4.3 Construction of the one-sided protocols.

We now show how to transform an ordinary multi-prover proof system into a one-sided multi-prover proof system. We assume without loss of generality that the proof system has exactly two provers, both of which are deterministic. We also assume that  $V$  runs for at most  $b(n)$  steps on an input of size  $n$ , and asks exactly  $m$  questions of each prover. First, we consider protocol ONE-SIDED-PARTIAL, given in Figure 6.3.

The following lemma formally states that the protocol is one-sided, and detects incorrect proofs with nonnegligible probability.

**Lemma 6.4** Let  $(P_1, P_2, V)$  be an interactive proof system for language  $L$ . If  $x \in L$ , and  $(P'_1, P'_2, V')$  execute protocol ONE-SIDED-PARTIAL( $P_1, P_2, V, x$ ), then  $V'$  will always output “good.”. If  $x \notin L$  and  $V'$  follows its protocol ONE-SIDED-PARTIAL( $P_1, P_2, V, x$ ), then for all provers  $\hat{P}_1, \hat{P}_2$ ,  $(\hat{P}_1, \hat{P}_2, V')$  will output “bad.” with probability at least  $1/6mp$ .

**Proof:** First, we show that  $V'$  will always output “good” when  $x \in L$ . If  $x \in L$ , then  $\text{accepting}(P_1, P_2, V, x, b)$  will be at least  $\frac{2}{3} \cdot 2^{b(n)}$ . Hence, by Lautman’s lemma, such a polynomially sized covering set  $T$  will indeed exist. By the definition of a covering set, and by the definition of the protocol, we have that for any  $R$ , more than half of the  $R_k$ ’s will be in  $\text{accepting}(P_1, P_2, V, x, b)$ . Thus, for any  $R$ , more than half of the transcripts output by  $P'_1$  will be accepting. Also,  $P'_2$ ’s answers will always be consistent with those given by  $P'_1$ , by definition of the protocol, so  $V'$  will always output “good.”

We now consider the case where  $x \notin L$ . We wish to bound above the probability that  $V'$  outputs “good.” We can assume without loss of generality that  $\hat{P}_1, \hat{P}_2$  are deterministic.

**Protocol One-Sided-Partial( $P_1, P_2, V, x$ )**

- 1: Let  $S$  be equal to  $\text{accepting}(P_1, P_2, V, x, b)$ , and let  $T$  be a covering set for  $S$ , where  $|T| \leq n^c$ . Here,  $c$  is a constant depending on  $P_1, P_2, V$ , and  $|x| = n$ . Prover  $P'_1$  sends  $T$  to  $V$ .
- 2: Verifier  $V'$  picks  $R \in \{0, 1\}^{b(n)}$ , and sends  $R$  to  $P'_1$ . Writing  $T = \{t_1, \dots, t_p\}$ , we define  $R[k] = R \oplus t_k$ , for  $k \in [1, p]$ . These strings are to thought of as possible random tapes of the original verifier  $V$ .
- 3: For each  $k \in [1, p]$ , prover  $P'_1$  computes the set of transcripts of the conversations  $V$  would have had with  $P_1, P_2$ , given random tape  $R[k]$ . Let  $q_j^i[k]$  be the  $j$ th question  $V$  asks  $P_i$ , using tape  $R[k]$ , and let  $a_j^i[k]$  be the  $j$ th answer  $V$  receives from  $P_i$ , using tape  $R[k]$ . Prover  $P'_1$  sends these transcripts, which we denote by  $\tau[1], \dots, \tau[p]$  to  $V'$ .
- 4: Verifier  $V'$  uniformly selects  $i \in [1, 2]$ , and  $j \in [1, m]$ . It also chooses  $k$  uniformly such that  $\tau[k]$  is an accepting transcript. Verifier  $V'$  send  $P'_2$  the values of  $q_1^i[k], \dots, q_j^i[k]$  and  $a_1^i[k], \dots, a_{j-1}^i[k]$
- 5: Prover  $P'_2$  sends  $a_j^i[k]$  to  $V'$ .
- 6: Verifier  $V'$  outputs either “good,” or “bad,” according to the following rules.
  - If no more than  $p/2$  of the transcripts were accepting, then  $V'$  outputs “bad.”
  - If any of the transcripts given to  $V'$  has  $V$  asking a question different from what it would ask given the conversation so far and  $V$ 's coin tosses, then  $V'$  outputs “bad.”
  - If  $P'_2$  gives anything but  $a_j^i[k]$  in Step 4,  $V'$  outputs “bad.”
  - Otherwise,  $V'$  outputs “good.”

Figure 6.3: Core protocol for creating one-sided interactive proofs.

**Protocol One-Sided-Complete( $P_1, P_2, V, x$ )**

1:  $P'_1, P'_2, V'$  execute protocol ONE-SIDED-PARTIAL( $P_1, P_2, V, x$ )  $6nmp$  times in series. If  $V'$  ever outputs “bad,” it aborts the protocol and rejects. Otherwise, it accepts.

Figure 6.4: Driver protocol for making protocols one-sided.

As with the proof of Theorem 6.1, let strategy  $S_i^*$  be defined as follows. On input  $([(q_1, a_1), \dots, (q_i, a_i)], q_{i+1})$ ,  $S_i^*$  outputs the answer given by  $\hat{P}_2$  when asked  $i, q_1^i[k], \dots, q_j^i[k]$  on input  $x$ .

By our definitions, we have that  $S = \text{accepting}(P_1^*, P_2^*, V, x, b)$  will be at most  $2^{b(n)}/3$ . Thus, for any set  $T$ , and a uniformly distributed  $R$ , the probability that  $|\text{hit}(S, T, r)| \geq |T|/2$  will be bounded above by  $2/3$ . Thus, with probability at least  $1/3$ , for no more than half of the  $R_i$ 's will  $\text{transcript}(P_1, P_2, V, R_i, x)$  be accepting. In such a situation, the only way in which  $V'$  can possibly output “good” is for  $\hat{P}_1$  to make at least one of the transcripts  $\tau[k]$  be accepting, and different from  $\text{transcript}(P_1, P_2, V, R_i, x)$ .

Now, if  $\tau[k]$  is an accepting transcript, then in Step 4 of the protocol, verifier  $V'$  will be select it for testing with probability at least  $\frac{1}{p}$ . If this happens, then the same argument as in the proof of Theorem 6.1, it will be detected with probability at least  $\frac{1}{2m}$  (the number of provers is equal to 2).

Combining all the conditional probabilities, the probability of the verifier outputting “bad” on input  $x \notin L$  is at least

$$\frac{1}{2} \cdot \frac{1}{p} \cdot \frac{1}{2m} = \frac{1}{6m}. \quad \blacksquare$$

Finally, it is easy to boost the verifier's chance of catching false proofs by running the protocol many times in series. The final protocol is as follows.

If  $x$  is in the language, then the verifier is guaranteed to accept all the runs, and is thus guaranteed to accept. If, on the other hand,  $x$  is not in the language, then the probability that  $\hat{P}_1, \hat{P}_2$  could fool the verifier  $6nmp$  times is at most

$$\left(1 - \frac{1}{6mp}\right)^{6nmp} \leq 2^{-\Theta(n)},$$

for  $n$  large. The theorem follows.  $\blacksquare$

## 6.5 Transforming 2-prover proof systems into zero-knowledge 2-prover proof systems.

### 6.5.1 Using a normal form for 2-prover protocols.

We can greatly facilitate the proof of our theorem by using the normal form for 2-prover proof systems established in the proof of our one-sidedness result.<sup>5</sup> Proof systems transformed according to Protocol *one-sided-partial* have the following general form:

1.  $P'_1$  sends  $V'$  a message,  $a_0$ .
2.  $V'$  sends  $P'_1$  a sequence of random coin tosses, whose length depends only on  $|x|$ .
3.  $P'_1$  sends  $V'$  a message,  $a_1$ .
4.  $V'$  picks a random sequence of bits,  $q_2$ , whose length depends only on  $|x|$ .
5. Based on the values of  $a_0, q_1, a_1, q_2$ , and  $x$ ,  $V'$  deterministically decides whether to abort the protocol. If not, the verifier deterministically computes  $Q, A_1$ .
6.  $V'$  sends  $Q$  to  $P'_2$ .
7.  $P'_2$  sends an answer,  $A_2$  to  $V'$ .  $V'$  accepts iff  $A_1 = A_2$ .

If  $x \in L$ , where  $L$  is the language accepted by the untransformed interactive proof system, then  $V'$  will always accept. Note that this implies that prover  $P'_2$  will be able to make  $A_2 = A_1$  for any values of  $q_1, q_2$ . There exists some constant  $c$ , such that for all sufficiently large  $x \notin L$ , and for all prover strategies,  $V'$  will reject with probability at least  $|x|^{-c}$ .

Note that it does not matter what  $P'_1$  learns after Step 3 of the protocol. Thus, in a certain sense, there is no need to hide anything from  $P_1$ , a fact we will exploit.

---

<sup>5</sup>Here is where our transformation ceases to be robust.

Other than these observations, We do not need to use any details about how  $a_0, q_1, a_1, q_2, Q, A_1$  are computed, except that all the verifier's computations are in probabilistic polynomial time. We say that a protocol in this form is in *one-sided normal form* with respect to language  $L$ .

### How the verifier receives information, and how to hide it.

To motivate our protocol, we first consider how a possibly cheating verifier may acquire knowledge. First, the verifier learns  $a_0, a_1, Q, A_1$ , which clearly has the potential to leak knowledge. Second, a cheating verifier may give the second prover a query  $Q'$ , that is different from  $Q$ , potentially getting an answer  $A_2$ , which is different from  $A_1$ . Seeing the answer to such an illicit question may give the verifier extra information.

We will use two different approaches to eliminating these sources of information. First, we will never allow the verifier to see  $a_0, a_1$ . this lack of knowledge still allows the verifier to compute  $q_1, q_2$ , which are random, but makes it impossible for the verifier to compute  $Q, A_1$ . This task must therefore be delegated to  $P_1$ , who is not trusted. We will perform this task by using our techniques for commital with zero-knowledge proofs.

The task of eliminating information about  $Q, A_1$  is somewhat subtler, since we must give the verifier some representation of  $Q, A_1$  in order for it to communicate productively with  $P_2$ . However, this representation must not convey knowledge. We also need to ensure that the verifier really does give this representation of  $Q$  to  $P_2$ . In order to do this, we use the unbreakable, unforgeable information-theoretic encryption scheme introduced in the last chapter.

### 6.5.2 A zero-knowledge protocol using envelopes with zero-knowledge proofs.

We now specify how to transform a protocol in the normal form given above into a protocol which is in zero-knowledge. This reduction will assume as primitive operations the ability to commit bits, and to give zero-knowledge proofs about the bits one has committed. It also assumes that the commital and zero-knowledge proof subprotocol may be simulated. Our protocol and simulator are given in Figures 6.5 and 6.6. For simplicity, we use the notation described above for naming the messages. That is,  $a_0$  is shorthand for the

first message given by  $P_1$  to  $V$ ,  $q_1$  is shorthand for the first question that  $V$  would have sent to  $P_1$ , etc. We assume without loss of generality that each string (e.g.  $a_0$ ) has a fixed length, as a function of  $|x|$ .

We wish to prove that our protocol transformation keeps the protocol a proof system, in some weak sense, and provides zero-knowledge. We require the following characteristics from our commital system with zero-knowledge proofs:

1. The second prover, even if malicious, gets no information during the commital and decommital protocols. More formally, the second prover's view depends only on the input  $x$ , his strategy, and whatever transpired in the preprocessing stage.
2. If  $P'_1$  commits a bit  $b$ , the verifier gets no knowledge (in the information theoretic sense) about  $b$ , except with probability less than  $1/|x|^c$ , for any constant  $c$ , and  $|x|$  sufficiently large. The simulator can simulate the commital of a bit without actually specifying any value for the bit. This simulation will be statistically indistinguishable from an actual bit commital.
3. The simulator can forge zero-knowledge proofs about forged bit commitments. Specifically, if a set of bits, whose commital was simulated by the simulator, there exists some setting such that the assertion about them would be true, then the simulated zero-knowledge proof will be statistically indistinguishable from an actual zero-knowledge proof of the assertion. As a corollary, giving a zero-knowledge proof reveals nothing about the committed bits (again in the information theoretic sense) than the fact that the assertion is true.
4. If provers  $\hat{P}_1, \hat{P}_2$  fail to properly commit a set of bits, or commits a set of bits and then attempt to prove a false assertion about these bits, they will be detected with some constant probability  $\rho > 0$ .

A more concrete way of viewing this abstraction is to imagine that the provers have envelopes in which they can place messages, and public notaries. At some point in the protocol, the public notaries will look inside these envelopes and certify that some true statement about them is indeed correct.

We use a very general description of our commital scheme because we do not need to know any of its details for our proof. Note, however, that

### Protocol 2-Prover-Zero-Knowledge( $P_1, P_2, V$ )

**Note:** We assume that  $(P_1, P_2, V)$  is in normal form, as described above. In the the description below,  $a_0, q_1, a_1, q_2, Q, A_1$  are generated according to the same distributions that would be generated by  $(P_1, P_2, V)$ . The party which will generate each of these values in the transformed protocol will have access to all the necessary information that the original party had.

- 0: Before the protocol has begun, provers  $P'_1$  and  $P'_2$  agree on a random prime  $p$ , whose length is equal to  $|x|^c$  for some fixed  $c$ . The constant  $c$  is chosen so that  $|p|$  will be more than twice the maximum possible length of  $Q, A_1$ , or  $x$ . Fixing  $p$ , they also agree on random encryption keys,  $K_1$  and  $K_2$ , (according to the encryption scheme presented in the last chapter) and on the value of  $a_0$ .
- 1:  $P'_1$  sends  $p$  to verifier  $V'$ , who verifies that it is prime.  $P_1$  commits  $a_0, K_1, K_2$  to the verifier.
- 2:  $V'$  computes  $q_1$ , and sends it to  $P'_1$ .
- 3:  $P'_1$  computes and commits  $a_1$ .
- 4:  $V'$  computes  $q_2$ , and sends it to  $P'_1$ .
- 5:  $P'_1$  computes  $A_1, Q$ , and sends  $K_1(Q), K_2(A_1)$  to  $V'$ , along with a zero-knowledge proof that this is what  $V$  would have computed, given the values of  $a_0, a_1, q_1, q_2, K_1, K_2$ . If  $V'$  is not convinced by the proof, it aborts the protocol.
- 6:  $V'$  sends  $K_1(Q)$  to  $P'_2$ .  $P'_2$  aborts if he gets something which is not a legitimate encryption (under key  $K_1$ ). Otherwise,  $P'_2$  inverts  $K_1$  to obtain  $Q$ , computes  $P_2$ 's response to this question,  $A_2$ , then sends back  $K_2(A_2)$ .  $V'$  accepts iff  $K_2(A_1) = K_2(A_2)$  (and hence iff  $A_1 = A_2$ ).

Figure 6.5: Protocol for making normal-form interactive protocols zero-knowledge.

**Simulator 2-Prover-Zero-Knowledge( $P_1, P_2, \hat{V}$ )**

- 1: The simulator generates and sends a prime  $p$  of the correct size to the possibly malicious verifier,  $\hat{V}$ . verifier. The simulator simulates the commital of  $a_0, K_1, K_2$  (note that it does not need to actually generate any such values).
- 2:  $\hat{V}$  computes some value,  $\hat{q}_1$ , and sends it to the simulator.
- 3: The simulator simulates the commital of  $a_1$ .
- 4:  $\hat{V}$  computes some value,  $\hat{q}_1$ , and sends it to the simulator.
- 5: The simulator picks  $\langle Q \rangle$  and  $\langle A_1 \rangle$  at random from  $Z_p$ , and sends them to  $\hat{V}$ . It then simulates the proof that these values would be  $K_1(Q), K_2(A_1)$  given the values of  $a_0, a_1, q_1, q_2, K_1, K_2$ .
- 6:  $\hat{V}$  sends some value,  $\langle \hat{Q} \rangle$  to the simulator (now simulating  $P'_2$ ). If  $\langle \hat{Q} \rangle \neq \langle Q \rangle$ , the simulator simulates  $P'_2$  aborting the protocol. Otherwise, it sends  $\langle A_1 \rangle$  to  $\hat{V}$ .

Figure 6.6: Simulator for protocol 2-Prover-Zero-Knowledge.

these properties of bit-committal with zero-knowledge proofs have appeared before: They are all achieved by our schemes based on oblivious transfer. Hence, if we can implement oblivious transfer in the 2-prover model, we can plug the committal protocols into our transformation. We will show how to do this in the next section, and thus implement our transformation without any assumptions.

We now show some key properties of our transformed protocols. Informally, we show that the probability of an honest verifier accepting an input  $x$  in the transformed protocol is essentially the same as its probability of accepting  $x$  in the original protocol. We also show that the transformed protocol will be zero-knowledge.

First, we make the simple observation that getting the verifier to accept some  $x \in L$  in the transformed protocol is no more difficult than in the original protocol.

**Lemma 6.5** Let  $(P_1, P_1, V)$  be a protocol in one-sided normal form with respect to language  $L$ . Let  $(P'_1, P'_2, V')$  be the transformed protocol, 2-PROVER-ZERO-KNOWLEDGE( $P_1, P_2, V$ ). Then for all input  $x \in L$ ,  $(P'_1, P'_2, V')$  will always accept.

**Proof:** There is a trivial mapping from conversations among  $P'_1, P'_2, V'$ , to conversations among  $P_1, P_2, V$ . Furthermore, a conversation in the transformed protocol is accepting iff the induced conversation in the original protocol is accepting. By the construction of the protocol, the values of  $a_0, q_1, a_1, q_2, Q, A$  will be generated according to the same distribution by  $(P'_1, P'_2, V')$  as they were by  $(P_1, P_2, V)$ . The lemma follows. ■

Similarly, we can show that convincing the verifier to accept a string  $x \notin L$  is as hard in the transformed protocol as in the original protocol.

**Lemma 6.6** Let  $(P_1, P_1, V)$  be a protocol in one-sided normal form with respect to language  $L$ . Let  $(P'_1, P'_2, V')$  be the transformed protocol, 2-PROVER-ZERO-KNOWLEDGE( $P_1, P_2, V$ ). There exists a constant  $c$  such that for  $|x|$  sufficiently large, and  $x \notin L$ ,

$$(\forall \hat{P}_1, \hat{P}_2) \text{accept}(\hat{P}_1, \hat{P}_2, V', x) < 1 - |x|^{-c}.$$

**Proof:** First, let us assume that prover  $\hat{P}_1, \hat{P}_2$  never try to violate the protocol for commitment with zero-knowledge proof, always commit legal encryption keys, and give genuinely prime numbers  $p$ . Under this assumption,

we claim that there is a mapping from  $(\hat{P}_1, \hat{P}_2)$  to  $(\hat{P}'_1, \hat{P}'_2)$  such that

$$\text{accept}(\hat{P}'_1, \hat{P}'_2, V, x) \geq \text{accept}(\hat{P}_1, \hat{P}_2, V', x).$$

Note that by the definition of our normal form,  $\text{accept}(\hat{P}_1, \hat{P}_2, V', x)$  will be bounded away from 1.

We specify  $\hat{P}'_1, \hat{P}'_2$  in terms of  $\hat{P}_1, \hat{P}_2$ . Conceptually,  $\hat{P}'_1$  and  $\hat{P}'_2$  will run a simulated conversation between  $\hat{P}_1, \hat{P}_2$  and  $V'$  that will correspond to their actual conversation with  $V$ .

1. On input  $x$ , during the preprocessing stage, provers  $\hat{P}'_1, \hat{P}'_2$  simulate  $\hat{P}_1, \hat{P}_2$  through Step 1 of the transformed protocol. At this stage,  $\hat{P}_1$  will have committed  $a_0, K_1, K_2$ , and sent  $V'$  the value of  $p$ , hence these values will be known to both  $\hat{P}'_1$  and  $\hat{P}'_2$ .  $\hat{P}'_1$  then sends  $V$  the value of  $a_0$ .
2. On receiving  $V$ 's first query,  $q_1$ ,  $\hat{P}'_1$  simulates  $\hat{P}_1$  on receiving this query. The simulated  $\hat{P}_1$  will commit an answer,  $a_1$ , which  $\hat{P}'_1$  will echo back to  $V$ . We note that at this point in the simulated protocol,  $V'$  would inevitably send  $\hat{P}_2$  the value of  $K_1(Q)$ , where  $Q$  is the query that  $V$  will send to  $\hat{P}'_2$ . This follows because the computation of  $Q$  is deterministic, and our assumption that  $\hat{P}_1$  will not attempt to cheat during the decommittal and zero-knowledge proof process. The state of  $\hat{P}_2$  after the end of this stage of the protocol may be simulated by running  $\hat{P}_2$  using a simulated view of the committal/decommittal protocols. This view can be simulated by  $\hat{P}'_2$  due to our assumptions about the committal scheme.
3. When  $V$  sends  $\hat{P}'_2$  its query,  $Q$ ,  $\hat{P}'_2$  continues the simulation of  $\hat{P}_2$ , sending it the value of  $K_1(Q)$  (note that  $\hat{P}'_2$  knows the value of  $K_1, K_2$ ). When  $\hat{P}_2$  outputs some answer,  $\hat{A}$ ,  $\hat{P}'_2$  sends  $V$  the value of  $K_2^{-1}(\hat{A})$ . This is always possible, since if  $K_2$  is a well-defined key, it defines a permutation.

It is not hard to see that  $V$  will accept if  $V'$  would have accepted in the simulated conversation, and hence the acceptance probability must be at least as large in the real conversation as in the simulated one. By the definition of one-sided normal form, there must be some  $c$  such that for  $|x|$  sufficiently large, and  $x \notin L$ , we have

$$(\forall \hat{P}'_1, \hat{P}'_2) \text{accept}(\hat{P}'_1, \hat{P}'_2, V, x) < 1 - |x|^{-c}.$$

By the above argument, the upper bound of  $1 - |x|^{-c}$  holds when  $\hat{P}_1, \hat{P}_2$  use the restricted strategy.

Now we consider the general case, where the provers  $\hat{P}_1$  and  $\hat{P}_2$  may attempt to violate the commitment protocols or give invalid values for  $K_1, K_2, p$ . We call such cheating “foolish” cheating, since it guarantees that the verifier will reject with probability at least some constant  $\rho > 0$ , by the specification of our commitment protocols. We now show that foolish cheating can not significantly decrease ones probability of detection. For notational convenience, we denote by  $foolish(\hat{P}_1, \hat{P}_2, x)$  the event that  $\hat{P}_1, \hat{P}_2$  cheat in a foolish fashion during the protocol, with verifier  $V'$  and input  $x$ . We denote by  $F(\hat{P}_1, \hat{P}_2, x)$  the probability that  $foolish(\hat{P}_1, \hat{P}_2, x)$  occurs. We can write the probability of  $V'$  accepting as

$$F(\hat{P}_1, \hat{P}_2, x) \cdot \text{prob}\{V' \text{ accepts} | foolish(\hat{P}_1, \hat{P}_2, x)\} + \\ (1 - F(\hat{P}_1, \hat{P}_2, x)) \cdot \text{prob}\{V' \text{ accepts} | \neg foolish(\hat{P}_1, \hat{P}_2, x)\}.$$

As a consequence of the inequalities shown above, this probability is bounded above by

$$F(\hat{P}_1, \hat{P}_2, x) \cdot (1 - \rho) + (1 - F(\hat{P}_1, \hat{P}_2, x)) \cdot (1 - |x|^{-c}) \leq 1 - |x|^{-c},$$

for  $x$  sufficiently large. The lemma follows. ■

A sensitive reader may object to our assumption that by simulating the two provers,  $\hat{P}_1, \hat{P}_2$ , one can determine the values of the bits that  $\hat{P}_1$  actually commits. This ability holds in most scenarios one can think of, such as the physical envelopes model. More importantly, it certainly holds in the protocols based on oblivious transfer, using the implementation of oblivious transfer discussed in the next section.

Finally, we show that the transformed protocol is indeed statistical zero-knowledge, regardless of the strategy of a possibly malicious verifier.

**Lemma 6.7** Let  $(P_1, P_2, V)$  be a protocol in one-sided normal form with respect to language  $L$ . Let  $(P'_1, P'_2, V')$  be the transformed protocol, 2-PROVER-ZERO-KNOWLEDGE( $P_1, P_2, V$ ). Then for all  $\hat{V}$ , the output of simulator 2-PROVER-ZERO-KNOWLEDGE( $P_1, P_2, \hat{V}$ ) will be statistically indistinguishable from the distribution on actual transcripts of  $(P'_1, P'_2, \hat{V})$ .

**Proof:** We essentially appeal to our assumption about the simulatability of the commital protocols, and to the unbreakability of the encryption scheme.

First, we note that up through Step 4 of the actual protocol, prover  $\hat{P}_1$  only commits bits. The simulator merely runs the commital simulator, which we hypothesize is statistically indistinguishable from a run of the actual commital scheme.

In Step 5 of the actual protocol, prover  $P'_1$  sends  $\hat{V}$  the values of  $K_1(Q), K_2(A_1)$ , along with a zero-knowledge proof that they are the proper encryptions. At this point in the simulation, the simulator sends  $\hat{V}$  two random elements of  $Z_p$ , along with a simulation of the zero-knowledge protocol. To see that these two events are statistically indistinguishable, we first note that  $K_1(Q), K_2(A_1)$  will be distributed independently and uniformly over  $Z_p$ . This follows immediately from the unforgeability properties of the encryption functions, and the fact that  $K_1, K_2$  are independently and uniformly distributed over all legal keys. By our hypothesis, the simulated zero-knowledge proof will be indistinguishable from an actual proof, provided that there is some setting of the committed variables for which the theorem is true. This condition is trivially satisfied, since there exists some values of  $K_1, K_2$  that will map a given number to a desired destination point.

Thus, the simulated interaction of  $\hat{V}$  with  $P'_1$  is indistinguishable from the actual interaction. We must now show that the remaining part of the simulation, namely the simulated interaction of  $\hat{V}$  with  $P'_2$ , is also indistinguishable from the actual interaction.

In Step 6 of the simulation, the verifier sends a message,  $\langle \hat{Q} \rangle$ , to the simulated  $P'_2$ . If  $\langle \hat{Q} \rangle = \langle Q \rangle$ , the simulator sends back  $\langle A \rangle$ , otherwise it simulates  $P'_2$  aborting the protocol. We argue that this behavior is indistinguishable from the actual behavior of the protocol, by a case analysis on the types of inputs that  $\hat{V}$  can give to  $P'_2$ . In the actual protocol,  $\hat{V}$  can send  $P'_1$  either

1. The value of  $K_1(Q)$ , which corresponds in the simulation to  $\langle Q \rangle$ .
2. Some value which is not a valid encryption, using key  $K_1$ .
3. Some value which is not  $K_1(Q)$ , but is a legitimate encryption, using key  $K_1$ .

In the first case, where  $\langle \hat{Q} \rangle = \langle Q \rangle$ , the simulation behaves exactly as the actual protocol would (noting that  $\langle Q \rangle$  corresponds to  $K_1(Q)$  and  $\langle A \rangle$  corresponds to  $K_2(A_1)$ ). In the second case, the simulation again

behaves exactly as the actual protocol would, since  $P'_2$  would indeed abort after seeing an illegal encryption. However, in the third case, the behavior of the simulation and the actual protocol are distinguishable. Essentially, the simulator does not have a third case - anything which isn't exactly the expected question is treated as garbage.

However, this discrepancy in the two behaviors is insignificant, for the same reason as with the CHAIN protocol. This is due to the unforgeability property of the encryption system. If the verifier gets no information about  $K_1, K_2$ , then no matter what strategy it uses, or how powerful it is, the probability that any new message it generates corresponds to a legitimate encryption is bounded above by  $2^{-n}$ . By the properties of the envelope scheme,  $\hat{V}$  has only a negligible probability of getting any information about  $K_1, K_2$ . Hence, the one case where the simulation breaks down happens with only negligible probability, and so the simulation is indistinguishable from the actual protocol. The lemma follows. ■

### 6.5.3 Running our protocols in series.

The properties we have proved for our zero-knowledge protocol transformation do not quite prove our main theorem, since the transformed protocol is not a proof system. However, as with our previous protocols, this deficiency may be remedied by serially running the protocol sufficiently many times in series. To simulate this protocol, it suffices to run the simulator many times in series. The fact that the malicious verifier  $\hat{V}$  has knowledge or previous runs of the protocol is immaterial: The proof of Lemma 6.7 holds regardless of the power or prior knowledge of  $\hat{V}$ .

## 6.6 Achieving Oblivious Transfer in the Multi-Prover Model.

### 6.6.1 Introduction.

In the previous section, we showed how to prove anything provable in zero-knowledge, modulo the existence of suitable a scheme for committing bits with zero-knowledge proofs of assertions about these bits. Naturally, we would like to eliminate this assumption. Since we can base such commitment

schemes on oblivious transfer, it suffices to exhibit a protocol for oblivious transfer in the two-prover model.<sup>6</sup> This section gives a simple implementation for oblivious transfer.

Our implementation of oblivious transfer will not be as ideal as we would like. For instance, if the provers are allowed to exchange any information after the conclusion of the protocol, the transfer will no longer be oblivious. Thus, the protocol we give can only be used in certain restricted situations. Fortunately, the use outlined in the previous section is one such situation.

It is in fact possible to implement oblivious transfer in a way which is partially resistant to interprover communication. This was first done in [BGKW], where we used the implementation to outline a robust zero-knowledge transformation for two-prover protocols. However, both the protocol for oblivious transfer, and the construction of the transformed protocol are somewhat complicated, and we do not go into this proof in this thesis.

### 6.6.2 Bit commital in the two-prover model.

In order to implement our oblivious transfer protocol, we first implement a form of bit-commital. In this protocol, we assume that provers  $P_1$  and  $P_2$  possess some bit  $b$ , which they wish to commit in advance. At some later time, either one of the provers may wish to decommit the bit. We assume This protocol is easy to implement in the two-prover model, provided that:

1. The provers are allowed to communicate before the protocol begins.
2. No communication of any kind is allowed between the two provers from the time of the commital to the time of the decommital. Our notion of noncommunication is very strict here: The messages sent from the verifier to prover  $P_i$  cannot depend on its conversation with prover  $P_{1-i}$ .

We present our commital and decommital protocols in Figures 6.7 and 6.8. They are, like most of our workhorse protocols, based on the classic exclusive-or trick. To weakly commit a bit,  $b$ , the provers randomly expand it as an exclusive-or of three bits,  $b_0 \oplus b_1 \oplus b_2$ . Clearly, allowing the verifier learn the value of two of these bits,  $b_i, b_j$ , will give him no information about  $b$ . Therefore, we simply have the verifier choose  $i, j$  at random, ask prover  $P_1$

---

<sup>6</sup>There are still a few technical details that will have to be addressed, as we will see.

**Protocol 2-Prover-Commit( $b, k$ )**

**0:** Beforehand, when the two provers can talk to each other, they choose a sequence of bits  $b_0, b_1, b_2$ , subject to

$$b = b_0 \oplus b_1 \oplus b_2.$$

**1:** The verifier uniformly picks  $q_1, q_2$ . He sends  $q_1$  to Prover 1, and  $q_2$  to Prover 2. Prover 1 sends the verifier the value of  $b_{q_1}$ , and Prover 2 sends the verifier the value of  $b_{q_2}$ . The verifier rejects if he is given inconsistent values.

Figure 6.7: Committing bits in the two-prover model.

**Protocol 2-Prover-Decommit( $b_0, b_1, b_2$ )**

**1:** The prover sends the verifier the values of  $b_0, b_1, b_2$ . The verifier rejects if any of the  $b_i$ 's are inconsistent with the values he received in the commital phase. Otherwise, the verifier reconstructs

$$b = b_0 \oplus b_1 \oplus b_2.$$

Figure 6.8: Deccommitting bits in the two-prover model.

for the value of  $b_i$ , and ask prover  $P_2$  for the value of  $b_j$ . A prover decommits  $b$  by sending the verifier  $b_0, b_1, b_2$ . If these bits disagree with either of the bits learned by the verifier during the commital stage, he aborts.

To argue that this is truly a commital protocol, we must show that the provers cannot, without some risk of detection, cause the verifier to decommit an incorrect value for a bit. For our argument, we can assume that the provers flip all their coins (perhaps infinitely many) before the beginning of the commital protocol, and then act deterministically based on their resulting state.

Define  $b_i^l$  to be the answer that would be given by prover  $P_l$  after being asked to reveal  $b_i$ . If, for any  $i$ ,  $b_i^1 \neq b_i^2$ , the verifier will reject with probability at least  $\frac{1}{9}$ . If prover  $P_l$ , during the decommital stage, gives a value

of  $b_i$  that is different from  $b_i^{3-l}$ , the verifier will reject with probability  $\frac{1}{3}$ . Thus, the provers must act consistently, both with each other at the time of commitment, and later on when they decommit.

Simulating the commital protocol is straightforward: One merely has to give random, consistent answers. Simulating the decommital of a bit is likewise very easy. After the commital phase, the value of at least one of the three bits is unspecified. Therefore, the simulator can pick the unspecified bits uniformly such that the total exclusive-or is equal to the value of the bit the simulator wants to decommit. This simulation is easily seen to be perfect. One might argue that the above protocols should be called “weak-commit” and “weak-decommit,” since they do not guarantee that cheating provers will be caught with probability exponentially close to 1. However, we do not need anything stronger for use in our interactive proofs, which only seek to catch a cheating prover with some nonnegligible probability. We can then run our interactive proofs many times in series to amplify their reliability.

### 6.6.3 Our transfer protocol.

We give our oblivious transfer protocol in Figure 6.9. Essentially, the provers randomly break their bit  $b$  into two bits,  $b_0$  and  $b_1$ , such that  $b = b_0 \oplus b_1$ , and commit these bits to the verifier. They then flip fair coins,  $r_1$  and  $r_2$ , with the verifier by the standard “flipping into the well” trick. Prover  $i$  then reveals the value of  $b_{r_i}$ .

We now want to claim that this protocol in some sense implements oblivious transfer. If the provers ever attempt to violate the commital or decommital protocols, they will be caught with nonnegligible probability. Therefore, for our purposes, we need only the case where the provers do not cheat during the commital/decommital process, but may try to cheat in other ways. First, if either the provers or the verifier are honest, then  $r_1, r_2$  will be distributed at random, so we may always assume that this is the case. Then no matter what strategy it uses, the verifier will get  $b$  iff  $r_1 \neq r_2$ , which will happen with probability  $\frac{1}{2}$ . Otherwise, it will get no information about  $b$ , in the information theoretic sense. If the verifier gives truly random values for  $q_1$  and  $q_2$ , then prover  $P_i$  will have no information about  $r_{3-i}$ , regardless of what values they committed. Thus, immediately after the conclusion of the protocol, neither prover will know whether the verifier received bit  $b$ .

**Protocol OT( $b$ )**

- 0:** During the preprocessing phase, the provers uniformly choose bits  $b_0, b_1, p_1, p_2$ .
- 1:** The provers and the verifier use protocol 2-PROVER-COMMIT to commit  $b_0, b_1, p_1, p_2$  to the verifier.
- 2:** The verifier uniformly chooses bits  $q_1, q_2$ , and for  $i \in [1, 2]$ , sends  $q_i$  to prover  $P_i$ . We define  $r_i = p_i \oplus q_i$ .
- 3:** For  $i \in [1, 2]$ , prover  $P_i$  uses protocol 2-PROVER-DECOMMIT to decommit  $p_i, b_{r_i}$ . In addition, prover  $P_1$  sends the verifier the value of  $b \oplus b_0 \oplus b_1$ . If the verifier detects any cheating in the commital or decommital protocols, then he aborts the protocol. If  $r_1 = r_2$ , then the verifier reconstructs nothing. Otherwise, it reconstructs  $b$  from the values  $b_0, b_1, b \oplus b_0 \oplus b_1$ .

Figure 6.9: Implementing oblivious transfer in the 2-prover model.

#### 6.6.4 Simulating the oblivious transfer protocol.

We would like to plug in our oblivious transfer protocol into all the protocols which we have already reduced to oblivious transfer. In order to ensure that the zero-knowledge properties of these protocols are preserved, we must come up with a simulator for our oblivious transfer protocol which is compatible with the requirements placed on it by the higher-level simulators.

All of the higher-level simulators can be thought of as working in the following manner: First, they randomly decide which of the bits being sent will actually get through (the mask), and then they decide on the values of the bits that are actually seen by the verifier. In the simulators described thus far, the simulator could simply say, "...bit 47 did not get through to the verifier, ...bit 48 was received with value 0,..." However, when we implement oblivious transfer using a protocol, we must be able to simulate the protocol with the three possible values (0, 1, #) for a bit being sent through to the verifier. This simulator is described in Figure 6.10.

**Remark:** Note that the simulator does not require the malicious verifier to speak to the provers in a synchronous fashion. A malicious verifier may

**Simulator OT( $b$ )**    */\*  $b \in \{0, 1, \#\}$  \*/*

- 0: The simulator uniformly chooses bits  $b_0, b_1, r_1$ . If  $b \in \{0, 1\}$  it computes  $r_2 = 1 - r_1$ , otherwise it sets  $r_2 = r_1$ .
- 1: The simulator fakes the commital of  $b_0, b_1, p_1, p_2$  (note that these values do not have to be specified at this time. To do this, he has each prover give random (though consistent) answers to the verifier's queries.
- 2-3: The verifier will send sends bits  $q_1, q_2$  to the simulated provers. When simulated prover  $P_i$  bit  $q_i$ , the simulator computes  $p_i = q_i \oplus r_i$ . The simulator then simulates  $P_i$  decommitting  $p_i, b_{r_i}$ . If  $b \in \{0, 1\}$  the simulator has the simulated  $P_1$  send  $b \oplus b_0 \oplus b_1$  to the verifier along with its decommitals. If  $b = \#$ , the simulated  $P_1$  sends a uniformly distributed bit.

Figure 6.10: Simulating the oblivious transfer protocol.

go along further in the conversation with one prover than with another. Indeed, the malicious verifier may, with one prover, continue on with another portion of the larger protocol, while still in the middle of the oblivious transfer protocol with the other prover.

This simulation is perfect, as shown in the following lemma.

**Lemma 6.8** The distribution output by simulator OT( $b$ ), where  $b \in \{0, 1\}$ , is the same as the distribution of protocol OT( $b$ ), conditioned on the verifier receiving the bit. The distribution output by simulator OT( $\#$ ) is the same as the distribution of OT( $b'$ ), conditioned on the verifier not receiving the bit, for  $b'$  equal to 0 or 1.

**Proof:** This protocol is finite, so in principle the lemma can be verified by an exhaustive case analysis. We give a more intuitive argument.

Recall that we say that the verifier receives a bit iff  $r_1 \neq r_2$ . First note that the distribution of the questions and answers in the commital schemes are identical in the simulated and actual protocols (i.e. the answers are random subject to consistency). The commital scheme gives no information about the committed bit, so the distribution of  $r_1, r_2$  will be uniformly distributed.

Since the comital step is simulated perfectly, the first query bit to a prover will be identically distributed in the simulated versus actual protocol. Let the first query message involve sending bit  $q_i$  to prover  $P_i$ . Choosing  $q_i, r_i$  according to the correct distribution, and then computing  $p_i$  is the same as choosing  $q_i, p_i$  according to the correct distribution, and then computing  $r_i$ . The decommital of  $p_i, b_{r_i}$  is also perfectly simulated. The same argument follows for the next query bit sent to a prover (note that the malicious verifier does not have to send  $q_1, q_2$  off at the same time, though doing so does not give him any advantage).

The one remaining point of difference in the simulated versus real protocol occurs when the verifier does not receive the bit. In the real protocol, prover  $P_1$  sends the verifier  $b \oplus b_0 \oplus b_1$ , and  $b_{r_1}$  (he also sends  $b_{r_2}$ , but in this case,  $r_1 = r_2$ ). The simulator sends a uniformly distributed bit, which we denote by  $b'$ , along with  $b_{r_1}$ . However, since  $b_{1-r_1}$  is uniformly distributed, and the verifier does not see it, the distributions of  $(b \oplus b_0 \oplus b_1, b_{r_1})$  and  $(b', b_{r_1})$  will appear identical to the verifier. ■

Now that we have implemented oblivious transfer, we can simply plug in our favorite commitment with zero-knowledge proof scheme based on oblivious transfer into our transformed zero-knowledge protocol. Whenever the verifier and  $P'_1$  need to perform an oblivious transfer, they run OT. The verifier forgets all about the details of the conversation it has, save for remembering whether it received the bit and the value of bits that were received.

### 6.6.5 When can we use the oblivious transfer protocol?

We need to understand exactly when our OT protocol can and cannot be used in larger protocols. We must address the logistics of actually using the protocol, and the problems with cheating on the part of the prover or verifier.

For our application, we initially use the second prover solely for implementing the OT protocol. Logistically, this gives some slight trouble in determining coordinating the second prover with the first prover and the verifier. We cannot expect the second prover to know how many oblivious transfers are to be executed. This is easily dealt with by having it set up to make the maximum number of oblivious transfers that could be needed by the protocol. It is a trivial exercise to pad our protocols with extra oblivious

transfers so that they always use the maximum number possible.

Our OT protocol behaves ideally with respect to the verifier. Therefore, any protocol which uses it in place of an ideal oblivious transfer protocol will remain zero-knowledge. However, it remains to be shown that plugging our OT protocol into a commitment scheme preserves the fact that it is a commitment scheme. The OT protocol's security with respect to the provers depends strongly on a lack of interprover communication. If either prover can communicate with the other prover any details about its conversation with the verifier, then the obliviousness argument given above breaks down. This severely hampers the use of the oblivious transfer protocol, but for our purposes it is still effective.

Let us consider our zero-knowledge protocol where we use a commitment with zero-knowledge proof scheme based on oblivious transfer. Such a scheme consists of a phase in which the verifier interacts with  $P'_1$ , and then sends  $P'_2$  a message. While it is interacting with  $P'_1$ , it only interacts with  $P'_2$  in order to execute the oblivious transfers necessary to run the commitment protocols.

As far as  $P'_1$  is concerned, the OT protocol is ideal in this initial phase, since prover  $P'_2$  cannot possibly leak any information to  $P'_1$ . However, we cannot make the same argument about  $P'_2$ . As soon as the verifier sends its message to  $P'_2$ , there is the possibility that this message was influenced by  $P'_1$  in order to help  $P'_2$  subvert the OT protocol. Therefore we must, and do, assume that  $P'_2$  learns which bits the verifier received in all the OT protocols up to that point.

This fact appears very ominous, but is in fact irrelevant to the security of the commitment scheme. Everything that is going to be decommitted already has been by this point. Therefore, it is irrelevant if  $P'_2$  has information that could be used to decommit a value of a bit different than the value actually committed.

### **Will our commitment schemes have the required properties?**

We can easily verify that commitment protocols using OT will obey all the properties required for the zero-knowledge protocol transformation given in the last section.

In protocol OT, the second prover's view consists solely of a set of random queries from the verifier. These queries are made completely independent of

anything else in the protocol. This fulfills the first condition.

Our commitment with zero-knowledge proof schemes that have been based on oblivious transfer all guarantee that the verifier gets information about a committed bit with only negligible property, and that the commital/decommital/zero-knowledge proof protocols can all be simulated. Thus, the second and third conditions are fulfilled.

Finally, the fourth condition is implied by the definition of our commital schemes, and the fact that any attempts to subvert the OT protocol will be detected with probability bounded away from 0. It should be noted that even if the commitment protocol based on oblivious transfer allowed for only a negligible chance of successful prover cheating, the resulting commitment protocols can only guarantee a bounded chance of prover cheating. This is because they could try to subvert the OT protocol. This will cause them to get caught with nonnegligible probability, but if they succeed even once, one must assume that this will allow them to subvert the protocol at large. However, this constant probability of catching cheating provers is all our previous lemma requires.

## References.

- [At] Atkin, A.O.L., to appear.
- [APR] Adleman, Pomerance, Rumely, "On distinguishing prime numbers from composite nubers," *Proceedings of the 21<sup>st</sup> FOCS*, IEEE, 1980, 387-406.
- [Ba] Barrington, David. "Bounded Width Polynomial Size Branching Programs Recognize Exactly Those Languages in  $NC^1$ ", *Proceedings of 18th STOC*, 1986, pp. 1-5.
- [Bl] Blum, M., "How to Prove a Theorem So No One Else Can Claim It: Zero Knowledge Proofs," ICM 1986.
- [Bl2] Blum M., "Three applications of oblivious transfer," manuscript.
- [Bo] Bosma, W. "Primality testing using elliptiv curves," report 85-12, Mathematisch Institut, University van Amsterdam, 1985.
- [BC] Brassard, Gilles and Claude Crépeau. "Zero-Knowledge Simulation of Boolean Circuits," *Proceedings of the 27<sup>th</sup> FOCS*, IEEE, 1986, 188-195.
- [BCC] Brassard, Gilles, Claude Crépeau, and David Chaum, "Minimum Disclosure Proofs of Knowledge," manuscript.
- [BCR] Brassard, Gilles, Claude Crépeau, and Jean-Marc Robert. "Information Theoretic Reductions Among Disclosure Problems," *Proceedings of the 27<sup>th</sup> FOCS*, IEEE, 1986, 168-173.
- [BG] Blum, Manuel, and Shafi Goldwasser, "An *Efficient* Public-Key Encryption Scheme Which Hides All Partial Information," manuscript.
- [BGGHKMR] Ben-Or, Michael, Oded Goldreich, Shafi Goldwasser, Johan Håstad, Joe Kilian, Silvio Micali, Phillip Rogaway, "IP is in Zero-Knowledge," *Proceedings, Advances in Cryptology*, Crypto 1988. (For an earlier appearance of this result, see [IY].)

- [BGKW] Ben-Or, Michael, Shafi Goldwasser, Joe Kilian, and Avi Wigderson, "Multi-Prover Interactive Proof Systems, Removing Intractibility Assumptions," These proceedings. *Proceedings of the 20<sup>th</sup> STOC*, ACM, 1988.
- [BGW] Ben-Or, M., S. Goldwasser, A. Wigderson. "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation." *Proc. of 20<sup>th</sup> STOC* (1988), 1-10.
- [BR] Ben-Or, M. and T. Rabin. "Verifiable Secret Sharing and Multiparty Protocols with Honest Majority." To appear, 21<sup>st</sup> STOC, ACM, 1989.
- [C] Crépeau Claude, "On the Equivalence of Two Types of Oblivious Transfer", *Crypto87*.
- [CC] Chudnovsky, D. V., and G. V. Chudnovsky, "Sequences of numbers generated by addition in formal groups and new primality and factorization tests," *Advances in Applied Mathematics*. 7 (1986).
- [CCD] D. Chaum, C. Crépeau and I. Damgard. "Multiparty unconditionally secure protocols," *Proceedings of the 20<sup>th</sup> STOC*, ACM, 1988.
- [CDvdG] Chaum, David, Ivan Damgard, and Jeroen van de Graaf. "Multiparty Computations Ensuring Secrecy of Each Party's Input and Correctness of the Output," *Proceedings of CRYPTO '87. Proceedings of CRYPTO '85*, Springer-Verlag, 1986, 477-488.
- [CL] Cohen, H., and H. Lenstra, "Primality testing and Jacobi sums," *Math. Comp.* 42 (1984).
- [Con] Condon, A. "Bounded Space Probabilistic Games," *Proceedings, 1988 Structure in Complexity Theory Conference*.
- [EGL] Even S., Goldreich O., and A. Lempel, *A Randomized Protocol for Signing Contracts*, CACM, vol. 28, no. 6, 1985, pp. 637-647.
- [GK] Goldwasser, Shafi and Joe Kilian, "Almost all primes can be certified," *Proceedings of the 18<sup>th</sup> STOC*, ACM, 1986. 316-329.

- [GMR] Goldwasser, Shafi, Silvio Micali, and Charles Rackoff. "The Knowledge Complexity of Interactive Proof-Systems," *Proceedings of the 17<sup>th</sup> STOC*, ACM, 1985, 291–304.
- [GMW] Goldreich, Oded, Silvio Micali, and Avi Wigderson. "Proofs that Yield Nothing but the Validity of the Assertion, and a Methodology of Cryptographic Protocol Design," *Proceedings of the 27<sup>th</sup> FOCS*, IEEE, 1986, 174–187.
- [GMW2] Goldreich, Oded, Silvio Micali, and Avi Wigderson. "How to play any mental game," *Proceedings of the 19<sup>th</sup> STOC*, ACM, 1987, 218–229.
- [GV] Goldreich, O., Vainish, R. "How to Solve any Protocol Problem: An Efficiency Improvement", *Crypto 87*.
- [HB] Heath-Brown D. R., "The differences between consecutive primes," *j. London Math. Soc.* (2), 18 (1978), 7–13.
- [ILL] Impagliazzo, Russell, Leonid Levin, and Mike Luby, "Pseudo-Random Generation from One-Way functions," *Proceedings of the 21<sup>th</sup> STOC*, ACM, 1989.
- [IY] Impagliazzo, Russell and Moti Yung, "Direct Minimum Knowledge Computations," *Proceedings, Advances in Cryptology*, *Crypto 1987*.
- [K] Kilian, Joe, "On The Power of Oblivious Transfer," *Proceedings of the 20<sup>th</sup> STOC*, ACM, 1988.
- [L] Lenstra, H., "factoring integers with elliptic curves," *Annals of Mathematics*, to appear.
- [LL] Lenstra, A. K., and H. W. Lenstra, "Algorithms in number theory," *University of Chicago Technical Report 87-008*.
- [M] Miller, Gary, "Riemann Hypothesis and a test for primality," *JCSS* 13 (1976), 300–317.
- [MO] Micali, Silvio, and Rafail Ostrovsky, personal communication.
- [MP] Maier H., Pomerance C., personal communication.

- [P] Pratt, "Every prime has a succinct certificate," SIAM J. of Comp. (1975), 214-220.
- [R] Rabin, M., "Probabilistic Algorithms for testing primality," J. of Num. Th. 12, 128-138 (1980).
- [R2] Rabin, M., "How to exchange secrets by oblivious transfer", Tech. Memo TR-81, Aiken Computation Laboratory, Harvard University, 1981.
- [Sch] Schoof, R., "Elliptic Curves over Finite Fields and the Computation of Square Roots mod  $p$ ," Math. Computation, Vol. 44, Num 170, April 1985.
- [Sil] Silverman, "The Arithmetic of Elliptic Curves," Springer-verlag, New York, 1986.
- [SS] Solovay and Strassen, "A fast Monte-Carlo test for Primality," SIAM. J. of Comp. 6 (1977).
- [T] Tate, "the arithmetic of Elliptic Curves," Inventiones Math. 234, (1974), 179-206.
- [Y] Yao, Andrew C. "Protocols for Secure Computations," *Proceedings of the 23<sup>rd</sup> FOCS*, IEEE, 1982, 160-164.
- [Y2] Yao, Andrew C. "How to Generate and Exchange Secrets," *Proceedings of the 27<sup>th</sup> FOCS*, IEEE, 1986, 162-167.

5325-9