Alternative Models for Zero Knowledge Interactive Proofs

by

Uriel Feige

Thesis for the Degree of
Doctor of Philosophy

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

1990

Abstract

An *interactive proof* system is a procedure by which a computational unlimited (but untrusted) prover can convince a polynomial time verifier of the validity of some complex mathematical statement. In the first part of our thesis we study a variation of the interactive proof model: Two provers, who cannot communicate between themselves, try to convince a constant space verifier of the validity of an assertion. We show that both in the case where the provers collaborate (either both cheat or both tell the truth), and in the case where the provers oppose each other (one tells the truth, the other one cheats, but the verifier does not know which is which), the verifier can verify membership in any recursively enumerable language. We use this result to show that computing the expected payoff of *reasonable games of incomplete information* is undecidable, thus solving a previously open question.

An interactive proof system is *zero knowledge* if the verifier cannot learn from it any mathematical fact other then the validity of the statement being proved – not even "why" this statement is true. This paradoxical concept turned out to be extremely useful in the design of cryptographic protocols. However, we demonstrate that it is lacking an important desirable property: Zero knowledge is not preserved under composition of protocols. This motivates the search for alternatives to the concept of zero knowledge. We introduce the concepts of *witness indistinguishability* and *witness hiding*, and develop their theory. Both these concepts are weaker than zero knowledge: They restrict only certain types of information that may be passed from prover to verifier in a proof system. Nevertheless, we show that witness indistinguishability and witness hiding can be successfully used in many cryptographic scenarios, and they may be superior to zero knowledge in applications which involve composition of protocols. In particular, we use these concepts to solve open questions related to the concept of zero knowledge: We show how polynomially many provers can share the same random reference string in giving polynomially many *noninteractive* zero knowledge proofs, and we design constant round zero knowledge *arguments* based on the sole assumption that one-way functions exist.

ACKNOWLEDGEMENTS

I would like to thank Adi Shamir, my thesis advisor, for his great help. Discussions with Adi helped many of the ideas in this thesis to evolve, and his observations and comments clarified and simplified most of its parts.

Oded Goldreich followed the development of some parts of this thesis, and his insightful remarks proved very valuable on several occasions.

During my studies at Weizmann, I also enjoyed working with David Peleg, Prabhakar Raghavan and Eli Upfal. Though work with them was not related to the subject of this thesis, I have benefited a lot from it, and the things I have learned from them undoubtedly influenced my whole work as a Phd student, and beyond.

Special thanks to my fellow students at the Weizmann Institute, Avital Schrift, Irad Yavneh, Moshe Tennenholtz, Muli Safra, Oded Maler, Rafi Heimann, and Yonatan Dym, for making each day at the office (and outside) a delightful social event.

Last, but most important, I would like to thank my wife Daphna, and my children Razi and Dror, for their love and support at home, and for putting up with a person whose head is filled with Zero Knowledge.

Contents

Chapter 1

Introduction

1.1 Interactive Versus Mathematical Proofs

It is a well known fact that four colors are sufficient to color any map such that no two countries that share a common border are colored with the same color. Geographers knew this fact for centuries, but Mathematicians proved its correctness only recently (in 1976). The proof produced for this statement is not an ordinary mathematical proof, one that any trained mathematician can verify. Rather, it involves a detailed case generation and analysis, which could be done only with the aid of a computer. Accepting the correctness of such a proof started a debate, since it involves asserting the correctness of software, but computer programs are known to have "bugs". It may be the case that as long as the computer program outputs "the four color theorem is false", its programmers continue to search for bugs in it, but once it outputs "the four color theorem is true", the programmers, who believe that this is the correct assertion, stop searching for bugs in the program. Given the current state of software verification, can we trust the output of such a program?

The notion of *interactive proofs* suggests a new approach for verifying the correctness of statements such as the four color theorem. The idea is to treat the computer which proves the theorem as a blackbox, without trying to formally verify the correctness of its software and the reliability of its hardware. The computationally powerful computer will have to convince the computationally limited human verifier that the four color theorem is true. The computer would be programmed (hopefully without bugs) to follow a prespecified verification protocol. At the end of the protocol, the verifier will perform by himself some simple computations, and decide by their result whether to accept the four color theorem, or to reject the computer as "buggy".

What would the ingredients of a verification protocol be? Certainly, the computer would not do all the talking, as this would amount to the computer writing down a short mathematical proof of the four color theorem, and we cannot expect the computer to find such a proof by itself (no offense meant against the AI community). So the verifier must also actively participate in the protocol, by guiding the computer with questions. Thus one ingredient of the protocol is interaction – it would be an *interactive proof.* Another ingredient which is likely to be required is randomization. If the verifier's questions are computed in a deterministic way, then the whole transcript of the protocol when it ends must again constitute a short mathematical proof of the four color theorem. So the verifier must use randomization in the construction of his questions. This also implies that when the protocol ends, the verifier would have only gathered statistical evidence to the correctness of the four color theorem. But unlike the evidence that we have now, this evidence will have *controllable error probability.*

We presented the two ingredients of interactive proofs: Interaction and randomization. But are these two ingredients sufficient? It is not obvious how these ingredients can be used in order to construct a verification procedure with controllable error probability for the four color theorem. Indeed, when the concept of interactive proofs was introduced (1985), no one appreciated its full power. Only recently (1989) it was realized that these proof systems can be used to verify extremely complicated statements. It remains to be seen whether the general techniques now known for constructing interactive proofs can be used to verify the correctness of the four color theorem.

1.2 Zero Knowledge Proofs and the Story of Wilhelm Tell

Hans was very excited as he entered his little village in the Swiss Alps.

"I just met the legendary Wilhelm Tell!" he announced.

"Well, that's interesting" said Gans, somewhat doubtfully.

"Yes, I did. I saw him place an apple over his child's head, and shoot right through it using his bow and arrow. No one else but the real Wilhelm Tell could be sure not to hit the child by chance."

"I still find your story quite fantastic" Gans insisted.

"I knew you wouldn't believe me, and so I brought conclusive evidence." Hans triumphantly showed Gans the very apple that Wilhelm Tell used, with the arrow still stuck through it.

“I understand that you really believe that you saw Wilhelm Tell, but for me, an apple and an arrow do not constitute convincing evidence” said Gans.

“But certainly, you must accept this as evidence”, Hans replied with conviction. “Don’t be so suspicious”, he added, placing the apple over his own head.

“If you call that evidence”, said Gans as he picked an apple and stuck it on an arrow, “then I too have seen Wilhelm Tell!”

This little story serves to illustrate the essence of *zero knowledge* proofs. Hans regarded the apple with the arrow stuck through it as convincing evidence that he had met Wilhelm Tell, because he saw with his own eyes the fantastic circumstances by which the arrow entered the apple. But Gans, who only saw the end result, could not be impressed, because he himself could easily stick apples on arrows and produce the same end result. Likewise, the verifier in a zero knowledge proof is convinced in the truth of the statement being proved, because he witnesses the construction of the proof online. But he cannot use the transcript of the zero knowledge proof in order to convince anyone else in the truth of the statement, since there is a simple way to construct similar transcripts offline.

A written mathematical proof of a nontrivial statement cannot be zero knowledge, since the verifier can show it to others and convince them as well. In order to be zero knowledge, the proof must have an element of interaction: The verifier must be able to witness the order in which different parts of the proof are constructed. Thus the concept of zero knowledge is another motivation for considering interactive proofs.

1.3 Overview of This Chapter

In the next section we establish notation and formally define interactive proofs, zero knowledge and related concepts. In section 5 we present a historical overview of the development of these concepts, and of their use in the field of cryptography. In section 6 we give examples of two zero knowledge interactive proof systems. These protocols will also be used in later chapters as building blocks of more complex protocols. In section 7 we overview variations made to the standard interactive proof model, such as multiple prover systems and space bounded verifiers. Readers with good background in zero knowledge interactive proofs may choose to skip all these sections and read only section 8, which explains the goals of our research and the results obtained.

1.4 Notation and Definitions

1.4.1 Models of Computation

In this section we introduce the abstract machine model used throughout this Thesis.

Computing machines receive inputs and produce outputs. These inputs and outputs are encoded using a finite set of *letters*, denoted as the *alphabet* Σ. *Strings* over Σ are constructed by concatenating letters of Σ. Σ^n denotes the set of all strings of length n, and Σ^* denotes the set of all finite strings of arbitrary length (including zero). Unless otherwise stated, we assume that Σ consists of the three letters $\{0, 1, \#\}$, where the letter # is used only as a delimiter between logically different parts of a single string.

A *function* is a mapping from Σ^* (*arguments*, or *pre-images*) to Σ^* (*images*). A *predicate* is a mapping from Σ^* to $\{0, 1\}$. The arguments of a predicate which map to 1 are called *yes instances* of the predicate. The set of all yes instances of a predicate is often referred to as a *language*.

In general, there may be no simpler way to specify a function than by listing its argument-image relations in an infinite table. But we are interested in functions for which there is a procedure of computing the image from the argument. In this case, the computation procedure serves as a finite description of the function. The formal model by which we present computation procedures is the Turing machine.

A Turing machine has a *control* which can be in one of finitely many *states*. It has a oneway infinite *work tape*, and each cell of the work tape is capable of holding one letter at a time from a finite alphabet Σ. At any given point of time, a special read/write *head* points to a single cell which can be accessed. The input to the Turing machine is placed on a special *input tape* which has a read only head. A *configuration* of a Turing machine consists of the state it is at, the contents of its tapes and the location of its heads. The computation is regulated by a transition function δ and proceeds in steps. In a single computation step the machine may change the state of the finite control, read the contents of the cells under its read heads, write into the cells under its write heads, and move each head by one step either to the left or to the right. The computation begins with the finite control at a special initial state, and the heads placed at the leftmost cells of their respective tapes. The computation ends when the machine enters one of two special final states, the *accept* state or the *reject* state.

The language L_T that a Turing machine T accepts is the set of inputs on which T eventually halts in the *accept* state. The function f_T that T computes is defined for each input as the final contents of T's work tape when the computation ends. The value of f_T is not defined for arguments on which T does not halt.

In order to deal with interactive proofs, we extend the model of the Turing machine. A Turing machine is *probabilistic* if its transition function depends on the outcome of the toss of a fair coin (in addition to the state of the finite control and the contents of the cells under the read heads). A Turing machine is *interactive* if it has an additional one way *communication tape.* Two interactive machines share the same communication tape, where each machine in turn can read the last message of the other machine and write down a new message of its own.

Finally, we do not require interactive Turing machines to start computations on each input from scratch. We allow them to start with a nonempty worktape. The initial contents of the work tape are called *auxiliary input. Auxiliary advice* is auxiliary input which depends only upon the length of the input, but not on the specific input. Turing machines which compute with the aid of auxiliary advice are said to be *nonuniform.*

1.4.2 Complexity Classes

A *class* is a set of languages, predicates or functions. The class of all languages whose instances can be enumerated by some Turing machine is called the *recursively enumerable* (R.E.) languages. The class of all functions which can be computed by a Turing machine is called the *recursive functions.*

We are often interested in the amount of resources a Turing machine uses during a computation. The *space* of a Turing machine is the number of cells it accesses at least once on its worktape. Its *time* is the number of computation steps it makes. Both space and time are functions of the particular input and auxiliary input to the Turing machine. We are interested in *worst case* measures. If n denotes the length of the input, we express space and time as functions of n, where for each n we consider only the input of length n which on some auxiliary input requires maximal resources.

Ptime is the class of all languages accepted by polynomial time Turing machines (machines performing $O(n^c)$ computation steps, for some constant c). *Pspace* is the class of all languages accepted by polynomial space Turing machines.

A BPP-machine (bounded probability polynomial time) is a polynomial time probabilistic Turing machine which for any input either accepts with probability over 2/3, in which case the input is said to be accepted by the machine, or accepts with probability less than 1/3, in which case the input is said to be rejected. BPP is the class of languages accepted by BPP-Turing machines.

An NP-machine (nondeterministic polynomial time) is a polynomial time probabilistic machine. An input is said to be accepted by an NP machine M if the probability that M accepts is nonzero (i.e., there is a lucky *guess* which causes M to accept). The sequence of coin tosses which leads to acceptance on input x is called the *NP-witness* (or just *witness*) for x. NP is the class of all languages accepted by nondeterministic Turing machines.

A language L_1 *reduces* to language L if there exists a polynomial time algorithm A such that for any x, $A(x) \in L$ iff $x \in L_1$. If there exists a polynomial time algorithm for membership in L, then it can be used in conjunction with the algorithm A to determine membership in L_1. A language L is *NP-complete* if $L \in NP$ and any $L_1 \in NP$ reduces to L.

It is customary to define efficient (tractable) computations as those which can be done in polynomial time, possibly with the help of randomization. Thus the classes *Ptime* and BPP are considered tractable. The following relations are known:

Ptime $\subseteq$ *BPP* $\subseteq$ *Pspace.*

Ptime $\subseteq$ *NP* $\subseteq$ *Pspace.*

None of the containments is known to be strict, though all of them are conjectured to be strict. The relation between NP and BPP is not known, though it has been proven that if *NP* = *Ptime* then also *BPP* = *Ptime*. The NP-complete languages are assumed to intractable.

A probabilistic Turing machine is *expected polynomial time*, if for any input, its expected running time (over random coin flips) is polynomial.

Nonuniform-P is the class of functions which are computed by polynomial time Turing machines which have *auxiliary advice*. It is known that the class *nonuniform-P* is not extended by considering BPP machines with auxiliary advice.

The reader should be careful not to confuse language recognition issues with the power of the computing machines. For example, *BPP* $\subset$ *Pspace*, but a BPP-machine can do things that a *Pspace* machine cannot: It can produce random bits.

1.4.3 Interactive Proofs

An *interactive proof* is a two party protocol by which a *prover* P convinces a *verifier* V that a *common input* x of length n belongs to language L. Both prover and verifier are probabilistic interactive Turing machines. The protocol proceeds with each party receiving a message from the other party, performing probabilistic computations on its own, and then sending a message to the other party. Communication is allowed only through a special communication tape, and a machine cannot read the contents of any other tape of the other machine. The protocol ends when the verifier enters one of its final steps (*accept* or *reject*).

Throughout this thesis, $\nu(n)$ is a short hand notation indicating nonnegative functions which vanish faster than the inverse of any polynomial. For example, $f(n) \leq \nu(n)$ is a shorthand notation for:

$$\forall k \, \exists N \; s.t. \, \forall n > N \;\; 0 \leq f(n) < \frac{1}{n^k}$$

Negligible probability is probability behaving as $\nu(n)$. *Overwhelming* probability is probability behaving as $1 - \nu(n)$.

$A(x)$ denotes the output of a probabilistic algorithm A on input x. $V_{P(x)}(x)$ denotes V's output after interaction with P on common input x. Both outputs are random variables which depend on the outcomes of coin tosses.

Definition 1.4.1 *Let P denote an algorithm run by a probabilistic interactive Turing machine, and let V denote an algorithm run by a probabilistic polynomial time interactive Turing machine with no auxiliary input. Both machines share a common input tape and a communication tape. The pair (P, V) is an* interactive proof *system for language L if it satisfies:*

1. Completeness. *V accepts with overwhelming probability inputs which are in L. Formally:*

$$\forall x \in L \;\; \text{Prob}(V_{P(x)}(x)\text{accepts}) > 1 - \nu(n)$$

2. Soundness: *Even if the prover cheats and deviates arbitrarily from his prespecified protocol, the probability that V accepts an input which is not in L is negligible. Formally:*

$$\forall P' \;\; \forall x \notin L \;\; \text{Prob}(V_{P'(x)}(x)\text{accepts}) < \nu(n)$$

P' denotes a possibly cheating prover, where each new message that P' sends is computed (not necessarily effectively) as a function of all previous communications with V.

The probabilities are taken over the coin tosses of the participating parties: V, P and P'.

1.4.4 Indistinguishability

The concept of indistinguishability of random variables is central to modern cryptography. Informally, two distributions of strings are indistinguishable, if no observer can tell whether a randomly chosen string comes from one distribution or the other.

Let A be a probabilistic algorithm. Then $A(x)$ is a random variable. Let L be a language. Then $\{A(x)\}_{x \in L}$ is called an *ensemble* of random variables on L.

Definition 1.4.2 *The ensembles $\{A(x)\}$ and $\{B(x)\}$ are* statistically indistinguishable *on L if for all $x \in L$*

$$\sum_{\alpha \in \{0,1\}^*} |\text{prob}(A(x) = \alpha) - \text{prob}(B(x) = \alpha)| \; < \; \nu(n)$$

If the above sum equals 0 then the ensembles are perfectly indistinguishable.

Definition 1.4.3 *The ensembles $\{A(x)\}$ and $\{B(x)\}$ are* computationally indistinguishable *on L if for $x \in L$ and for any nonuniform polynomial time algorithm D*

$$|\text{prob}(D(A(x)) = 1) - \text{prob}(D(B(x)) = 1)| \; < \; \nu(n)$$

1.4.5 Zero Knowledge

Zero knowledge is a notion which has a paradoxical flavor: The verifier is convinced by the prover that indeed $x \in L$, without having any idea *why* this containment is true. From the verifier's point of view, the response of the prover in a zero knowledge proof is equivalent to a 1-bit response from a trusted oracle. Zero knowledge has proved to be a useful concept in the design of cryptographic protocols, as the prover can be forced to make only truthful statements, without being forced to reveal how these statements were derived.

Zero knowledge is a guarantee to the truthful prover that no information, other than $x \in L$, leaks during the interactive proof, even if the verifier cheats and deviates arbitrarily from his prespecified protocol. Informally, a protocol is zero knowledge if the verifier can simulate the whole protocol offline even without interacting with the real prover. The formal definition of zero knowledge involves a probabilistic simulator M which uses the description of V and its inputs in order to simulate in expected polynomial time the execution of the protocol (P, V). In our notation, $M(x, y; V)$ denotes algorithm M's output on inputs x and y, where M may use algorithm V as a (blackbox) subroutine. M may repeatedly set inputs to V (including auxiliary inputs and coin tosses), and observe V's outputs.

Definition 1.4.4 *Proof system* (P, V) *is* perfectly *(*statistically*) (*computationally*)* zero knowledge *on L if there exists a simulator M, such that for any probabilistic polynomial time V', for any $x \in L$, and any auxiliary input y to V', $M(x, y : V')$ runs in expected polynomial time, and the two ensembles $V'_{P(x)}(x, y)$ and $M(x, y; V')$ are perfectly (statistically) (computationally) indistinguishable.*

1.4.6 Proofs of Knowledge

In an interactive proof, the prover may be infinitely powerful. In cryptographic applications, this is an unrealistic assumption. It is much more acceptable to view the prover as probabilistic polynomial time with some secret auxiliary input which helps him follow the protocol. Typically, L is an NP language, and P "knows" (has placed as auxiliary input) an NP witness for the common input.

It turns out that in many cryptographic applications, one would like the success of an interactive proof not only to indicate that $x \in L$, but also that the prover actually "knows" an NP witness for this statement. To formally define knowledge in this context is quite a tricky business. It is not sufficient to consider only the contents of the auxiliary input tape as knowledge. One must also consider the way this information is encoded and interpreted. Without thorough knowledge of the actual algorithm that the prover is running, it is impossible to decide whether he can actually exhibit a witness. The definition of knowledge which overcomes these difficulties is based on the concept of *knowledge extractor*, which is an analogue to the concept of simulator used in definition 1.4.4 of zero knowledge.

Our model of computation is the probabilistic polynomial time interactive Turing machine. Each interactive machine has an auxiliary input tape. P's auxiliary input is denoted by w.

Definition 1.4.5 *Let R be a relation $\{(x, w)\}$ testable in polynomial time, where the lengths of x and w are polynomially related. For any x, its* witness set *$w(x)$ is the set of w such that $(x, w) \in R$.*

Clearly, $L = \{x \mid \exists w \; s.t. \; (x, w) \in R\}$ is an NP language.

Definition 1.4.6 *An* interactive proof of knowledge *system for relation R is a pair of probabilistic polynomial time algorithms (P, V) satisfying:*

1. Completeness:

$$\forall (x, w) \in R \;\; \text{Prob}(V_{P(x,w)}(x) \text{ accepts}) > 1 - \nu(n)$$

2. Soundness:

$$\exists M \; \forall P' \; \forall x \; \forall w'$$

$$|\text{Prob}(V_{P'(x,w')}(x) \text{ accepts}) \;\; - \;\; \text{Prob}(M(x, w'; P') \in w(x))| \;\; < \;\; \nu(n)$$

P' is restricted to polynomial time computations, and $M(x, w'; P')$ runs in expected polynomial time.

The probabilities are taken over the coin tosses of V, P, P' and M.

We explain the typical way in which the soundness condition is proved for proofs of knowledge, so that the reader can become more familiar with the subtleties of this definition. The definition fixes the algorithm and auxiliary input of the possibly cheating prover P', and then compares the probabilities of two events: That of the verifier accepting P's proof (probability taken over the coin tosses of V and P'), and that of the knowledge extractor M computing a witness for the common input x (probability taken over the coin tosses of M). The knowledge extractor knows V's algorithm, and can use P' as a blackbox, feeding it with the relevant inputs. Though it makes sense to allow M to see w', the auxiliary input for P', we shall never use this in practice, and in effect we treat w' as part of the blackbox P'.

For cheating P', we do not know in advance what the value of *Prob*($V_{P'}$ *accepts*) is. Thus the knowledge extraction procedure starts with M simulating one random execution of (P', V). If V rejects, M halts. If V accepts, M repeatedly uses P' as a blackbox, and halts only after M extracts a witness for x (this explanation is inexact, as there are other conditions under which M halts, but they occur with negligible probability). This ensures that M's probability of extracting a witness is equal (upto negligible additive terms) to V's acceptance probability. In typical constructions of proofs of knowledge, it turns out that the running time of M in the latter case is inversely proportional to the acceptance probability of V. The case that V's acceptance probability is very small severely effects the *worst case* running time of M, but does not effect M's *expected running time*, hence this is the complexity measure that we use for M.

Observe that if $w(x)$ is empty, the definition implies that the probability that V accepts is negligible.

Definition 1.4.4 of zero knowledge, with minor modifications, applies also to proofs of knowledge.

Definition 1.4.7 *Proof of knowledge system (P, V) is* perfectly (statistically) (computationally) zero knowledge *on R if there exists a simulator M, such that for any probabilistic polynomial time V', for any $(x, w) \in R$, and any auxiliary input y to V', $M(x, y : V')$ runs in expected polynomial time, and the two ensembles $V'_{P(x,w)}(x, y)$ and $M(x, y; V')$ are perfectly (statistically) (computationally) indistinguishable.*

1.4.7 Cryptographic Background

In this section we survey some cryptographic primitives that serve us through most of this thesis.

The concept of intractability used in complexity theory is not strong enough to support complexity based cryptography. A function may be intractable even though it is easy to compute for almost all inputs. In order to have cryptographic applications, we need problems that are easy to generate, but hard to solve on almost all the generated cases. This is captured by the notion of a *oneway function.*

Definition 1.4.8 *A function f is* oneway *if for any x, $f(x)$ can be computed in polynomial time, but for random x, the probability of any nonuniform polynomial time algorithm of finding a preimage of $f(x)$ is negligible.*

It is not known whether oneway functions exist. In particular, if P=NP they do not exist. But the existence of oneway functions seems to be the minimal complexity *assumption* on which we can base cryptography. In fact, attempts to formulate seemingly weaker assumptions, such as the existence of functions with are hard to invert only on a nonnegligible fraction of the instances, or the ability to generate hard solved instances of NP problems, were found to imply the existence of oneway functions.

Definition 1.4.9 *A* pseudorandom bit generator *is a polynomial time deterministic algorithm which extends n-bit* seeds *to $2n$-bit images, such that the ensemble of images on uni-*

formly distributed random seeds is computationally indistinguishable from the ensemble of $2n$-bit uniformly chosen truly random strings.

Thus a pseudorandom generator "amplifies" randomness: From n random bits it generates $2n$ bits which seem random to any polynomial time observer. It is known that pseudorandom bit generators exist if and only if oneway functions exist ([BM84], [ILL89], [H90]).

A useful primitive in the construction of zero knowledge protocols is that of a *bit commitment.* Informally, it may be viewed as a protocol in which party A places the value of a single bit in a sealed envelope. Party B does not know the value of this bit, but can learn it by asking A to open the envelope.

Definition 1.4.10 *A* bit commitment *protocol is a two party protocol which proceeds in two phases: A* commitment *phase, and a* revealing *phase. After the commitment phase parties A and B share a* reference word *c which represents a bit b to which A is committed. After the revealing stage B learns the value of b. The protocol must satisfy:*

1. Completeness: *A can commit to either* $'0'$ *or* $'1'$.
2. Soundness: *After the commit stage, the probability that A can open the reference word both as* $'0'$ *and as* $'1'$ *is negligible.*
3. Security: *Let q be the probability that B guesses correctly the value of b without participating in the commitment protocol. Then before the reveal stage, the probability that B guesses correctly the value of b is at most* $q + \nu(n)$.

It is known that the existence of pseudorandom bit generators (and thus of oneway functions) implies the existence of bit commitment protocols [Naor89].

A particular example of a function which is (currently) conjectured to be oneway is that of exponentiation modulo a prime. This number theoretic function also has certain algebraic properties which makes it a convenient choice in many cryptographic applications.

Let p be a prime integer and let g be a generator of its multiplicative group Z_p^* (i.e., successive powers of g modulo p generate all the integers between 1 and $p-1$). Let c be a succinct certificate for p's primality and for g being a generator (e.g. see [Pratt]). Given (p, g, x), one can compute in polynomial time $y = g^x (mod\ p)$. The inverse operation, that of extracting the *discrete logarithm* modulo primes, is assumed to be intractable, even if c (a certificate for primality) is given in addition to p and g.

Definition 1.4.11 The Intractability of the Discrete Log Assumption (DLA): *There exists a random polynomial time algorithm which generates tuples* (p, g, c, x, y) *where p is* *prime, g is a generator, c is a certificate for p and g, and $y = g^x(\text{mod } p)$, such that any nonuniform polynomial algorithm has only negligible probability of outputing x on input (p, g, c, y).*

Remark: The discrete log problem is *random self reducible* [AL83]. This implies that given (p, g, c), either computing the discrete log is easy for all y, or it is easy only for a negligible fraction of all y.

1.5 Historical Overview

The purpose of this section is to briefly review the "state of the art" in terms of the topics of this thesis. It does not give a comprehensive bibliography of works in this field, and it ignores important work which is not directly relevant to the understanding of this thesis. The interested reader will find plenty of additional references by consulting the references cited here.

The concepts of *interactive proofs* and *zero knowledge* were first introduced in 1985 by Goldwasser, Micali and Rackoff [GMR85]. For some years, interactive proofs were believed to only slightly extend the complexity classes NP and BPP, with *graph nonisomorphism* being the most cited example of a language possessing an interactive proof but not known to lie in either of these classes [GMW86]. A surprising recent development ([N89], [LFKN89], [S89]) established that in fact interactive proof systems can test exactly those languages which are in Pspace. The protocol which achieves this has *perfect completeness* (i.e., with truthful prover and verifier it always succeeds) and uses only *public coins* (the prover sees all the coin tosses of the verifier).

Goldwasser, Micali and Rackoff [GMR85] demonstrated perfect zero knowledge proofs for certain number theoretic languages. It was soon recognized that under the assumption that *bit commitment* schemes exist (see Definition 1.4.10), all NP languages have computational zero knowledge proofs ([GMW86], [BCC88]). It is conjectured that perfect zero knowledge proofs do not exist for *NP-complete* languages [F87]. Zero knowledge was also established as a basic primitive for cryptographic protocol design ([GMW86], [GMW87]), and became the basis of practical identification and signature schemes [FS86].

The concept of *proof of knowledge* was implicit in the works of [GMR85], [GMW86] and others, and achieved formalization in [FFS87]. Definition 1.4.6 in this thesis is based on a later definition given in [TW87], which is easier to work with.

1.6 Two examples

1.6.1 Discrete Logarithm

We describe a simple zero knowledge interactive proof of knowledge of the discrete logarithm modulo a prime. This protocol was discovered by many researchers, with [TW87] being a convenient reference.

Common input: (p, g, c, x), where p is prime, g a generator of Z_p^*, c is a certificate for p's primality and for g being a generator, and $x \in Z_p^*$.

The prover's auxiliary input: w, the *discrete log* of x, satisfying $g^w = x \ (mod\ p)$.

The full protocol is obtained by repeating the following basic step $\log p$ times with independent coin tosses. The basic step is composed of three *moves*:

Protocol 1.6.12 *1. P chooses at random $r \in Z_p^*$, and sends $y = g^r \pmod p$ to V.*

2. V replies with a random bit b. (If V replies by any value other than 0/1, P halts.)

3. P sends z, and V accepts if $g^z = x^b y \pmod p$.

V accepts the whole protocol if it accepts each one of its basic steps.

We argue informally that this protocol is a complete, sound and perfect zero knowledge proof of knowledge of the discrete log.

Completeness: If $b = 0$, P can cause V to accept move 3 by sending $z = r$. If $b = 1$, P can cause V to accept by sending $z = r + w \pmod{p-1}$. Thus a truthful P, who knows w, can always follow this protocol.

Soundness: We construct a *knowledge extractor* M which has complete control over P, and which can extract w from P without prior knowledge of w. Fix r_P, an arbitrary sequence of random coin tosses for P. M randomly selects r_V, a sequence of $\log p$ coin tosses (as used by V). M simulates the system (P, V), treating the possibly cheating P as a black box which M feeds with the bits of r_V. In the end of the simulation, M evaluates V's acceptance condition. If V does not accept P's proof, M stops. Otherwise, M repeatedly executes the whole process $\log p$ times, where at execution j, M sends in move 2 of basic step j the complement of the j^{th} bit of r_V.

If for some j, V accepts basic step j of execution j, then M holds (y, z_0, z_1) satisfying $g^{z_0} = y \pmod p$ and $g^{z_1} = xy \pmod p$. This implies that $z_1 - z_0 = w \pmod{p-1}$, and so M extracts w.

Clearly, the running time of M is polynomial. We show that for any value of r_P:

$$|Prob(V_P \ accepts) \ - \ Prob(M(P) = w)| \ \leq \ 2^{-\log p}$$

where the probabilities are taken over the value of r_V. The only case where the first simulated execution succeeds and all subsequent executions fail to produce w, is when there is exactly one value of r_V on which V accepts. In this case, $Prob(V_P \ accepts) = 2^{-\log p}$, and the above equation holds with equality. In all other cases, $Prob(V_P \ accepts) = Prob(M(P) = w)$.

Remark: In the unlikely event that M fails to produce w, M may opt to determine w by exhaustive search. This is not a polynomial time procedure, but its contribution to the *expected* running time is only polynomial, due to the negligible probability of having it invoked.

Zero knowledge: We construct a simulator M which without knowledge of w simulates the whole protocol. M has complete control over the possibly cheating V. The complete simulation is constructed by performing the following simulation for each basic step.

M guesses a bit b' and chooses a random $r \in Z_p^*$. M sends $y = g^r/x^{b'}$ to V. If V's challenge at move 2 is $b = b'$, then at move 3 M sends $z = r$ and continues to the next basic step. It is easy to see that in this case the test of move 3 is satisfied, both when $b = 0$ and when $b = 1$. If $b \neq b'$, than M restarts the simulation of this basic step with fresh b' and r.

A subtle point is M's behavior in case cheating V's challenge is neither 0 nor 1. Unconditionally stopping the simulation in this case would result in a bias towards simulated executions which are stopped prematurely. The correct procedure is to stop the protocol with probability which equals the conditional probability of completing the simulation of the basic step on a legal challenge. This can be achieved if M stops the simulation only if $b' = 0$, but restarts the simulation of the basic step if $b' = 1$.

Note that regardless of the value of b', the distribution of the messages y that M sends at move 1 is the same as the distribution of messages that truthful P (who knows w) sends. Thus V's reply in move 2 is independent of the value of b'. Consequently, the expected number of tries until a single basic step is simulated is 2, and the whole simulation procedure takes expected time proportional to $2 \log p \ time(V)$ (where $time(V)$ denotes the running time of V, which is assumed to be polynomial). Likewise, the probability that a basic step is eventually simulated with $b = 1$ (or $b = 0$) is equal to the probability of this happening in a real execution of the protocol. It follows that the output of the simulation procedure is perfectly indistinguishable from a true execution of the protocol, even if V cheats and tries to deviate from his prespecified protocol.

1.6.2 Hamiltonicity

Zero knowledge interactive proofs of knowledge for *NP-complete* languages are especially valuable, since any other NP statement can be reduced to an instance of such a language. The protocol we present here, which is a modification of a protocol presented by Blum [Blum86], is due to Lapidot and Shamir [LS90]. It assumes the existence of secure bit commitment schemes (see definition 1.4.10).

Common input: G, the adjacency matrix of an n node directed graph.

The prover's auxiliary input: w, a directed Hamilonian cycle in G.

The full protocol is obtained by repeating the following basic step n times with independent coin tosses.

Protocol 1.6.13 *1. P chooses a random cycle on n nodes independently of the cycle $w \subset G$, and constructs its adjacency matrix H. P sends to V a secure commitment to each of the entries of H.*

2. V replies with a random bit b. (If V replies by any value other than 0/1, P halts.)

3. If $b = 0$, P reveals all commitments and V accepts if H indeed represents a single directed Hamiltonian cycle. If $b = 1$, P sends to V a permutation π that maps H onto the cycle $w \subset G$, and reveals the values of all entries in $\pi(H)$ which do not correspond to edges in G. V accepts if all these values are '0' (nonedges).

V accepts the whole protocol if it accepts each one of its basic steps.

We argue informally that this protocol is a complete, sound and computational zero knowledge proof of knowledge of Hamiltonicity.

Completeness: If $b = 0$, P has no problem following the protocol. If $b = 1$ then P, which knows both the cycle in H and the cycle $w \in G$, can find in polynomial time a permutation which maps one cycle to the other. Since now all the original '1' entries in $\pi(H)$ correspond to edges of G, all the bits that P reveals are necessarily '0', and V accepts.

Soundness: By the soundness property of the commitment scheme, even a cheating P can reveal each committed bit only in a unique way. The ability of P to perform move 3 when $b = 0$ implies that H contains a Hamiltonian cycle. The ability of P to perform move 3 when $b = 1$ implies that P knows a permutation which maps the cycle in H to edges in

G. From this it is easy to determine a cycle in G. Using the above observation, we can construct a knowledge extractor M which extracts a cycle in G from P. The construction of M is similar to the one given for Protocol 1.6.12 and is omitted.

Zero knowledge: We construct a simulator M which without knowledge of w simulates the whole protocol. M has complete control over the possibly cheating V. The complete simulation, which is based on ideas used in the simulation of Protocol 1.6.12, is constructed by performing the following simulation for each basic step.

M guesses a bit b'.

- If $b' = 0$, M proceeds as the real prover does, by choosing a random cycle and committing to its adjacency matrix. If V sends $b = 0$ at move 2, then M can complete move 3.
- If $b' = 1$, M commits to the adjacency matrix of the empty n node graph E (all entries are $'0'$). If V sends $b = 1$ at move 2, then M sends a random permutation π to V, and reveals all entries which correspond to nonedges of G in $\pi(E)$. Necessarily all these entries correspond to $'0'$s.

If $b \neq b'$, than M restarts the simulation of this basic step with fresh b'.

We have to show that the output of the simulation procedure is computationally indistinguishable from a true execution of the protocol, even if V cheats and tries to deviate from his prespecified protocol. The proof is similar to the proof that Protocol 1.6.12 is zero knowledge. The new problematic point in the proof is the following: The real prover always commits to Hamiltonian matrices, whereas the simulator sometimes commits to empty matrices. It has to be shown that these cases are computationally indistinguishable. The proof, which relies on the well known “hybrid” technique of [Yao82] and [GM84], is too lengthy to be given here.

Remark: The basic step of Protocol 1.6.13 is described as a sequence of three moves. The accuracy of this description depends upon the particular commitment scheme used. If it is the one constructed in [Naor89] based on any one-way function, then move 1 should be broken into two submoves, and consequently the basic step takes four moves. If the commitment scheme is based on one-way one-to-one functions (e.g., see [GL89]) then commitments can be performed in a single move, and the basic step indeed takes three moves.

1.7 Alternative Models

In the interactive proof model of [GMR85] the prover is infinitely powerful and the verifier is bounded to random polynomial time computations. The study of proof systems in which the verifier is space bounded rather than time bounded was initiated by Condon and Ladner ([C87], [CL86]), who proved that a log-space verifier can verify any *EXP-time* language. Dwork and Stockmeyer [DS88] extended this result to the case of finite state verifiers. They also defined the concept of zero knowledge for such systems, showing separation results between zero knowledge and non-zero knowledge proof systems with finite state verifiers. Condon and Lipton [CL89] prove a double-exponential space upper bound on the complexity of languages recognized by log-space verifiers. They also prove that if unending computations are allowed when the prover is cheating, a finite state verifier can accept any recursively enumerable statement (see also [L89]).

The study of *multiple prover* systems was initiated by Ben-Or, Goldwasser, Kilian and Wigderson [BGKW88]. They showed that two collaborating provers which are not allowed to communicate with each other can prove in perfect zero knowledge any NP statement to a polynomial time verifier. The power of these systems was fully understood only recently, when Babai, Fortnow and Lund [BFL90] proved that these systems accept any non-deterministic exponential time language.

A somewhat different model of multiple provers was suggested by Feige, Shamir and Tennenholtz [FST88]. In their model several opposing provers interact with a single verifier. At least one of the provers (unknown to the verifier) tries to help the verifier reach the correct decision, while the other provers try to fool him. It was shown that if all communication channels are public, then the language recognition power of such a system is exactly *P-space*. It is not known whether statements beyond *P-space* can be verified in such a system with private communication channels.

The concept of zero knowledge has also been modified in several ways. Blum, Feldman and Micali [BFM88] suggested the model of *noninteractive zero knowledge* in which the prover writes down the zero knowledge proof, and anyone can verify it against a universal publicly available random string (such as the RAND string of one million random digits). They showed (under a number theoretic assumption) that any NP language has a noninteractive zero knowledge proof system. Lapidot and Shamir [LS90] showed how to construct such proofs under the general assumption that one-way permutations exist. Noninteractive zero knowledge has become an important primitive for cryptographic protocols, with applications such as signature schemes [BG89] and encryption schemes [NY90].

Feige, Fiat and Shamir [FFS87] suggested the concept of *transferable information*, which is a weaker concept than zero knowledge. They showed that the parallel version of the Fiat-Shamir identification protocol does not reveal transferable information (i.e., the verifier cannot later represent himself as the prover) even though it is not known to be zero knowledge.

1.8 Overview of Thesis

In Chapter 2 we study the model in which two provers who cannot communicate between themselves prove to a constant space verifier that a common input x belongs to a prespecified language L. We show that both in the case where the provers collaborate and in the case where the provers oppose each other the verifier can accept any recursively enumerable language. By using our techniques we show that computing the expected payoff of reasonable games of incomplete information is undecidable, thus solving an open problem posed by Reif [Reif79]. We combine our techniques with the result of [BFL90] to show that in the multi-provers model [BGKW88] a constant space verifier can accept any nondeterministic exponential time language in polynomial time.

In Chapter 3 we take a closer look at the definition of zero knowledge, concentrating on the fact that the verifier is only polynomial time, whereas the simulator is *expected* polynomial time, and thus potentially stronger than the verifier. We discuss the difficulties which evolve if we try to make them both polynomial time, or if we try to make them both expected polynomial time.

In Chapter 4 we introduce the problem which motivates our search for alternatives to the concept of zero knowledge. We show that when two zero knowledge protocols are carried out in parallel, the verifier can use information from one protocol in order to extract the prover's witness in the other protocol. Goldreich and Krawczyk [GKr] proved a similar result, but only for the case that the provers are exponentially powerful. Our result applies also to the cryptographically interesting case in which the provers are polynomial time with auxiliary input.

A two party protocol in which party A uses one of several witnesses to an NP assertion is *witness indistinguishable* if party B cannot tell which witness party A is actually using. In Chapter 5 we define this concept and show that, unlike zero knowledge protocols, witness indistinguishability is preserved under arbitrary composition of protocols, including parallel execution.

In Chapter 6 we apply the concept of witness indistinguishability to solve an open problem associated with noninteractive zero knowledge, stated by Blum, De Santis, Micali, and Persiano: How to enable polynomially many provers to prove in writing polynomially many statements based on a single random reference string.

In Chapter 7 we introduce the concept of *witness hiding*, which informally means that by the end of such a protocol the verifier cannot compute any new witness to the statement being proven. We prove that if a statement has at least two independent witnesses, and none of them can be computed by the verifier before the protocol begins, then any witness indistinguishable protocol for this statement is also witness hiding. We discuss the possibility of using witness hiding as an alternative to zero knowledge, and show the relevance of this concept to *identification schemes*.

Zero knowledge arguments are zero knowledge proofs in which the soundness condition is required to hold only with respect to provers limited to random polynomial time computations ([BCC88], [BCY89]). In Chapter 8 we use the concept of witness hiding to construct constant round zero knowledge arguments to any NP statement under the sole assumption that one-way functions exist. Other known constructions of constant round zero knowledge arguments require much stronger cryptographic assumptions ([GKa], [BCY89]).

In Chapter 9 we extend the main result of Chapter 7 and prove that if a statement has at least two independent witnesses, then any witness indistinguishable protocol for this statement is also witness hiding (regardless of the value of the a-priori probability that the verifier can compute a witness to the input statement on his own).

Chapter 2

Multiple Provers and Constant Space Verifiers

2.1 Introduction

The notion of interactive proofs, introduced by Goldwasser, Micali, and Rackoff [GMR85] (*IP*) and by Babai [B85] (*AM*), was generalized to models with many provers in two different ways:

1. The multi-prover model (*MIP* model) of Ben-or, Goldwasser, Kilian and Wigderson [BGKW88]. The provers cooperate with each other, and either try to jointly help the verifier, or to jointly mislead him.
2. The noisy oracle model (*NO* model) of Feige, Shamir and Tennenholtz [FST88]. The provers oppose each other, with at least one prover trying to help the verifier and the others trying to mislead him.

In this chapter we use the terms "oracle" and "prover" interchangeably.

We investigate the language recognition power of constant space probabilistic verifiers in multi-oracle models. The power of space bounded verifiers in single prover models has been extensively studied. Condon and Ladner [CL86] and Condon [C87] give a general setting for studying the power of log-space verifiers in single prover models, and show the counter intuitive result that languages in exp-time can be accepted. Condon and Lipton [CL89] prove an upper bound of double-exponential space on the complexity of languages accepted by such systems. Simultaneous poly-time and log-space bounds on verifiers are studied by

Fortnow [F89] who shows that in case of public coins they can accept only languages in P, and by Rompel [R88] who shows that in case of private coins this model is as strong as IP. Dwork and Stockmeyer [DS88] study the case of finite state verifiers. The zero knowledge aspects of space bounded protocols (which are not discussed in this thesis), are studied in [DS88] and by Kilian [K88].

We know of no work investigating the power of space bounded verifiers in models with many provers. Still, work on games of incomplete information by John Reif [Reif84] and work on "Multiple-Person Alternation" by Peterson and Reif [PR79] are relevant to our work, and their techniques serve as an excellent starting point for developing protocols in our models.

Summary of our main results in this chapter:

- We construct a protocol by which a finite state verifier simulates a Turing machine (and thus tests any recursively enumerable statement), both in the opposing provers model and in the collaborating provers model. If any prover cheats during the simulation, the verifier detects this with high probability. Furthermore, the simulation stops with probability 1 whenever the simulated Turing machine itself stops.
- We show a connection with game theory, concluding that computing the payoff of optimal probabilistic strategies for reasonable two player games of incomplete information is undecidable. This solves an open question raised by Reif [Reif79] in 1979.
- Combining our techniques with the recent results of [BFL90] we show that in the collaborating provers model, a finite state verifier can verify in polynomial time any nondeterministic exponential time language.

Lipton ([L89], [CL89]) has recently constructed a protocol by which even a single prover can prove any r.e. statement to a finite state verifier, provided the soundness condition is relaxed to allow cheating provers to extend the execution of the protocol indefinitely. This implies an alternative proof to some of the results in this chapter, and in particular, the undecidability result for the opposing provers model. However, statements which require time $t(n)$ on a Turing machine require time $2^{2^{t(n)}}$ in Lipton's protocol, making it impossible to use Lipton's protocol in order to derive our results on simultaneous time and space bounds.

Most of the results of this Chapter (which predated Lipton's paper) appeared in [FS89a].

2.2 The Formal Model and Terminology

The verifier V is modeled as a probabilistic finite automaton (pfa). V has special *send* and *receive* states through which he communicates with the oracles. One pair of states is dedicated to each oracle, and V can send or receive only one character at a time. Thus V can communicate with each oracle in constant space, though he cannot remember the complete history of the communication. In contrast, each oracle may remember the complete history of his own communication with V, and use it in order to compute the next character it sends.

Randomness is incorporated into V by having the transitions of V depend not only on the state of its finite control and on the characters accessed by its read heads, but on the result of a flip of an unbiased coin as well.

The oracles may be either good or bad. Truthful oracles are modeled as computationally unbounded random interactive Turing machines. Cheating oracles are modeled as (possibly incomputable) functions from the history of their communication with V to the next character they produce. V does not know a-priori which are the good oracles and which are the bad ones.

A protocol is said to be *synchronous* if V's communications with the oracles occur at fixed points of time, with respect to some global clock. Otherwise, the protocol is said to be *asynchronous*. In asynchronous protocols, the oracles cannot derive any conclusions from the delay between successive attempts to access them. All the protocols we construct can be implemented in the synchronous model.

The probability of success of a protocol is defined in a natural way over the random tosses of V and of the good oracles. Bad oracles do not toss coins. Instead they nondeterministically choose the function which gives them the best chance of cheating.

If only one of the oracles is present, we have the IP model of GMR [GMR85] with space bounded verifiers. We denote this model as $IP(|)$, where the $|$ stands for one prover. We are interested in cases where at least two oracles are present. We distinguish between two submodels:

1. Multi-Prover (Ben-or, Goldwasser, Kilian and Wigderson [BGKW88]). The oracles collaborate in the sense that either all of them are good or all of them are bad. No oracle knows the communication of other oracles with V. We denote this model by $IP(\|)$, where $\|$ stands for collaborating oracles.

2. Noisy Oracle (Feige, Shamir and Tennenholtz [FST88]). The oracles oppose each other in the sense that at least one of them is good, and the others may be bad. V does not

know which is the good oracle. No oracle knows the communication of other oracles with V. We denote this model by $IP(\times)$, where $\times$ stands for opposing oracles.

We will in general consider cases where only two provers are present. It is a simple matter to adapt our protocols to cases where there are more than two oracles.

V's goal is to verify membership of the common input x of length n in the language L. Two conditions must hold:

1. Completeness: if $x \in L$, the probability that V accepts is greater than 2/3. (In the $IP(\|)$ model this is required only if all oracles are good. In the $IP(\times)$ model, this is always required, as one oracle is always good.)
2. Soundness: if $x \notin L$, then no matter what the strategy of the bad oracles is, the probability that V rejects is larger than 2/3.

When we consider recursively enumerable languages, we have to allow for nonhalting computations. In this case (and only in this case), we use the following relaxed soundness condition: If $x \notin L$, then no matter what the strategy of the bad oracles is, the probability that V accepts is smaller than 1/3.

2.3 Simulating a Turing machine

When dealing with time bounded verifiers, the role of the oracles is to actively help the verifier to save computation time (by suggesting good options when a nondeterministic choice is required, by giving counter-examples to wrong conjectures the verifier has, etc.). The key to understanding models with space bounded verifiers is to realize that the verifier's problem is not how to save time (which now he has plenty), but rather to provide him with space. In doing so, the oracles can assume a passive role of serving as work tapes. The verifier must be active in checking that the provers do their job correctly. Once storage $s(n)$ is reliably implemented, the verifier can check by himself statements of space complexity $s(n)$.

In this section we show how a finite state verifier can use two oracles as a reliable oneway-infinite work tape. V asks the provers to store values for him, and can later retrieve these values. The storage is reliable in the sense that when V retrieves a value from the tape, he can detect with high probability whether this value is the value V initially wrote, or whether the oracles changed this value without authorization. Furthermore, in case some mishandling of storage is encountered, V knows to which of the oracles to attribute the fault. This is a necessary requirement in the opposing provers model, because in this case the provers contradict each other to begin with, and so the fact that someone (without knowing who) mishandles the storage does not really teach V anything new.

We note that the finite control of any Turing machine can be encoded in the description of the finite state verifier. The location of the read/write head on the work tape can be encoded as a special symbol in the appropriate location on the work tape itself, and can be detected by scanning the nonblank portion of the work tape. Thus once V has an infinite work tape, he obtains the full power of a Turing machine.

Simple attempts to implement reliable storage fail. For example, V may try to store duplicate copies of each value, one with each prover. When V later retrieves the value, he checks that what the provers send match. This scheme offers V no advantages. In case of opposing provers, if a mismatch occurs, V does not know which one of the two provers cheated. In case of collaborating provers, nothing prevents them from deciding beforehand on a common policy of which values are to be changed and in what way. An equally naive scheme is to detect errors in retrieved values by using an error detecting code. This scheme would detect with high probability that an indifferent oracle returns a wrong value. But we are dealing with powerful cheaters which can easily create false values which pass the error detecting tests.

A storage policy which gives a partial solution to our problem is the following (adapted from Peterson and Reif [PR79]): V keeps two copies of the infinite tape, one with P_1, the other with P_2. Any access to memory involves reading all contents of the tape from both provers and retrieving (or replacing in case of a write operation) the value which follows the special character used to designate the location of the read/write head. Before the protocol begins, V randomly and secretly chooses one of the following two policies:

1. Check that both provers are giving the same values as contents of memory by alternating turns between them.

2. Run P_1 one access to memory ahead of P_2, again alternating turns between the provers. V checks that the contents of memory the provers send match, except for the last value which V wrote, which is updated with P_1 and not yet updated with P_2. (V always remembers in his finite memory what the last update was).

In order to cheat successfully, the provers must both cheat at the same time. Otherwise V detects a mismatch. But in order to synchronize the first attempt to cheat, the provers must guess whether one prover is running one access to memory ahead of the other or not. They have probability 1/2 of guessing wrong.

The above idea has two drawbacks:

1. The protocol detects cheating, but gives no indication as to which one of the two provers is cheating. Thus, it is worthless in the opposing provers model.
2. The protocol is inherently asynchronous. If the provers have a synchronized clock, they can cheat. (e. g. start sending wrong values at noon).

Theorem 2.3.14 *Both in $IP(\|)$ and in $IP(\times)$, a probabilistic finite state verifier can implement a oneway-infinite work-tape reliably.*

Proof: In our construction V "signs" the contents of each cell on the work tape, so that a cheating oracle who tries to change the contents of a cell must produce a new "valid signature" as well. Cell contents are chained in order to prevent duplication or reordering of validly signed cells. In order to defy attempts by the infinitely powerful oracles to break the signature scheme, we make it secure in an information theoretic sense. This is achieved by dividing the signature between the two oracles, such that no single one of them has enough information in order to forge a signature.

Let Σ be the alphabet of the infinite tape to be implemented, where Σ includes the blank character ω and a special character h to be placed on the current location of the read/write head. Denote the contents of the 1-way infinite work tape by m_1, m_2, m_3, m_4 etc., where $m_i \in \Sigma$. Encode each character as an integer in the range $[0, q-1]$, where q is a large enough prime satisfying $q \geq |\Sigma|$.

V chooses secretly and independently three random integers: $a, b, r_0 \in [0, q-1]$. For each location i on the work tape, in addition to m_i, V chooses a random value $r_i \in [0, q-1]$ and computes a signature s_i, where:

$$s_i = a \cdot m_i + b \cdot r_i + r_{i-1} \ \ (mod\ q)$$

Each prover receives from V only some of the values, without seeing what the other prover receives. P_1 receives m_i, r_i and s_i for all the odd i, P_2 receives them for all the even i. Thus P_1 holds m_1, r_1, s_1; P_2 holds m_2, r_2, s_2; P_1 holds m_3, r_3, s_3 etc, and V holds a, b, r_0. The computation of a signature involves a and b which are secretly held by V, r_{i-1} which was chosen randomly by V and held by the other prover, and r_i which is chosen randomly by V and held by the current prover.

In order to use his virtual work tape, V scans its contents from left to right by asking the provers to send him each time the appropriate m_i, r_i, s_i values. V checks each time that the signatures are valid. If an invalid signature is detected, V declares the prover sending the signature as cheater. Each time the contents of the work tape have to be updated, V chooses fresh values for all r_i, including r_0.

A formal description of V's algorithm follows. $x \leftarrow \$$ is used to denote that the value of x is randomly and uniformly selected as an integer in the range $[0, q-1]$. j is a variable which accepts the values 1 or 2, indicating which of the two provers is being addressed. $\bar{j}$ indicates the other prover (i.e., if $j = 1$ then $\bar{j} = 2$, and vice versa). Recall that ω denotes the blank symbol and h the read/write head.

Program 1: Initialization of work tape with contents of input tape.

```
$a \leftarrow \$$ ; $b \leftarrow \$$ ; $r_0 \leftarrow \$$ ; $r_3 := r_0$ ; $j := 1$ ; $m := h$ ;
repeat
    {
    $r \leftarrow \$$
    $s = a \cdot m + b \cdot r + r_3 \pmod{q}$
    send $(m, r, s)$ to $P_j$
    if $m = \omega$ then stop
    else
        {
        $r_3 := r$
        $j := \bar{j}$
        $m :=$ next character from input tape
        }
    }
```

Each prover stores the triplets (m, r, s) in the order in which he receives them.

Program 2: To be done on each access to the simulated work tape.

```
$j := 1$ ; $r_0 := q$ ; $m_2 := \omega$ ; $firstchar := true$ ;
repeat
    {
    receive $(m, r, s)$ from $P_j$
    $m_j := m$ ; $r_j := r$ ; $s_j := s$ ;
    if $s_j \neq a \cdot m_j + b \cdot r_j + r_{\bar{j}} \pmod{q}$
    then declare $P_j$ as cheater and stop
    else
```

```
        {
        $r \leftarrow \$$
        $if\ firstchar = true\ then$ { $r_0 := r$ ; $firstchar := false$ }
        $else$
            {
            compute new value for $m_{\bar{j}}$ from $(m_3, m_{\bar{j}}, m_j)$ and store in $m$
            $s = a \cdot m + b \cdot r + r_3 \ \ (mod\ q)$
            send $(m, r, s)$ to $P_{\bar{j}}$
            $if\ m = \omega\ then$ stop
            }
        $m_3 := m_{\bar{j}}$ ; $r_3 := r_{\bar{j}}$ ; $j := \bar{j}$ ;
        }
    }
```

Each prover sends the triplets (m, r, s) in the order in which they are stored, and replaces the last values he sends by the new values that V sends him.

Claim: The first attempt to cheat is detected by the verifier with probability $\frac{q-1}{q}$.

The proof of this claim involves two simple steps:

1. Up to the first time a prover tries to cheat, all values that P_1 (P_2, respectively) receive are independent of a, b:

   (a) m_i – Derived from the computation.

   (b) r_i – Chosen at random in the range $[0, q-1]$.

   (c) s_i – In one-to-one correspondence with r_{i-1}, which was chosen at random.

   Thus neither P_1 nor P_2 knows anything about a or b.

2. Assume that the first attempt to cheat involved cell i held by P_1. Thus V gave P_1 the values m_i, r_i and s_i, and P_1 later sent V the values M_i, R_i and S_i, at least one of which is different from the original value. If the cheating is not caught, the following equation must hold:

   $$S_i = a \cdot M_i + b \cdot R_i + r_{i-1} \ \ (mod\ q)$$

   In addition we know that

$$s_i = a \cdot m_i + b \cdot r_i + r_{i-1} \ \ (mod\ q)$$

Subtracting the two equations we get the following nontrivial relation between a and b:

$$(S_i - s_i) = (M_i - m_i) \cdot a + (R_i - r_i) \cdot b \ \ (mod\ q)$$

As q is prime, exactly q pairs (a, b) satisfy any nontrivial linear equation. But there are q^2 possible pairs (a, b). So the probability that P_1 who is ignorant of the values of (a, b) will cheat and still pass the signature test is only $\frac{1}{q}$.

This completes the proof of the claim, and the proof of the Theorem follows. $\diamond$

Now that we know how to implement infinite work-tapes reliably, it is not difficult to prove the following Theorem:

Theorem 2.3.15 *Any r.e. language has both $IP(\times)$ interactive proofs and $IP(\|)$ interactive proofs where the verifier is a pfa.*

Proof: Let L be an r.e. language and let T be the deterministic Turing machine accepting L. In order to verify that input x is in L, V simulates T's computation on x. T's finite control can be encoded into V. By Theorem 2.3.14, using two provers V can implement work tapes with controllable probability of error $(\frac{1}{q})$. If no error is detected and V's simulation of T's computation leads to T accepting x, V accepts x. If an error is detected, then V declares the prover responsible for the error as cheater. In the $IP(\times)$ model, this leads to acceptance of the claim of the other prover. In the $IP(\|)$ model, this leads to rejection of both provers. $\diamond$

The converse of the above Theorem is also true:

Observation 2.3.16 *Testing whether a pfa verifier accepts input x in an $IP(\times)$ or $IP(\|)$ proof system is in r.e..*

Proof: Given input x, we want to know if V accepts x. If V does accept x, there is some time threshold t by which V has accumulated probability greater than $\frac{1}{3}$ of accepting x.

In $IP(\times)$ it is sufficient to guess the probabilistic Turing machine implementing the good prover, and to guess t. Now one has to try out all possible functions of how the bad prover replies to each possible history shorter than t, and check that for each one, the probability of V accepting, taken over the tosses of V and the good prover, is greater than $\frac{1}{3}$.

In $IP(\|)$ model, the optimal provers are deterministic. Thus it is sufficient to nondeterministically guess t and two functions of how each prover replies to each possible history shorter than t, and check over all sequences of V's coin tosses shorter than t that V has probability greater than $\frac{1}{3}$ of accepting. $\diamond$

If the simulated Turing machine T does not halt on input x, then V does not halt on x either. This is unavoidable, because if V did halt and reject all inputs that T does not halt on, then techniques similar to those of the proof of Observation 2.3.16 would imply a recursive decision procedure for the *halting problem*. But if T does halt on input x, it is desirable that V halt as well. With the protocols presented above, this is not the case. A cheating prover has small probability $\frac{1}{q}$ of successfully diverting V's simulation (by changing one symbol in some intermediate configuration) to a nonhalting computation of T, and so V might not halt even though T does. It is possible to modify our protocol so that V halts with probability 1 on any input on which T halts. (This proof technique was independently discovered by Condon and Lipton [CL89].)

Theorem 2.3.17 *Any recursive language has both $IP(\|)$ and $IP(\times)$ verification systems where the verifier is a pfa which stops with probability 1.*

Proof: As in Theorem 2.3.15, the verifier simulates the computation of T. In order to stop, the verifier tosses a coin after each step he makes. If the coin comes up "Heads", the verifier continues with the simulation. If the coin comes up "Tails", the verifier aborts the current run and restarts the protocol from the beginning (with new random and independent a, b and r_0).

With probability 1, any infinite sequence of coin tosses contains infinitely many long enough sequences of "Heads" which can bring the protocol to its completion. If a prover tries to bring about an infinite computation, he must cheat every time this happens (otherwise V stops because T stops). The probability of not getting caught in infinitely many independent attempts to cheat is 0. $\diamond$

Note that we pay a high price for stopping with probability 1:

1. The expected running time of the protocol is exponentially higher than the running time of T.

2. The original protocols could be implemented with a oneway probabilistic finite automaton. It was sufficient that the verifier goes over the input tape just once, and copies its contents to the virtual work-tape. But now, when we need the possibility to restart the protocol, the verifier must have a two-way read head on its input tape.

Both problems can be avoided in the opposing provers model. In this case, whenever the protocol described in the proof of Theorem 2.3.14 exceeds the number of steps necessary to simulate T, the good prover can deliberately stop the protocol (and get blamed by V as a cheater). This is justified because the event that the protocol runs for too long without a lie being detected has small probability which does not significantly lower the good prover's probability of winning.

2.4 Games of imperfect information

In this section we describe an interesting application of our result on the power of $IP(\times)$ to game theory. This section does not deal with the $IP(\|)$ model.

Lemma 2.4.18 *Even a finite state verifier who cannot toss coins can verify any r.e. statement (provided the good prover can toss coins).*

Proof: The verifier's protocol uses random numbers in the range 0 to $q-1$. These numbers can be generated by asking each prover for a secret random number, and adding the two replies modulo q. Because at least one of the provers is good, the resulting number will be random, and the bad prover cannot know what its value is. $\diamond$

Our treatment of the verification system viewed the verifier as the active center of the system, and used provers only as memory. A complementary view is one in which the provers are two players actively playing a game, and the verifier is the passive "board" on which the game is played. The local state of the verifier serves as the position of the game. The messages the provers send serve as their moves. The transition table of the verifier as a deterministic finite automaton together with the contents of the input tape serve as the function computing the new position resulting from the previous position and the last move. The game is of imperfect information in the sense that the communication is secret, and the players (provers) cannot see the whole position (messages of the other prover). The game is "reasonable" in the sense that the size of the board (i.e. the memory of the verifier) does not change during the game. A strategy for a player is a (probabilistic) function from the observable history of a game to his next move. For exact definitions of the above terms see [Reif84].

In [Reif79] and [Reif84], Reif studies such reasonable games of incomplete information. He shows that determining whether one of the players has a strategy that *always* wins is complete for exp-exp-time. An open question raised in [Reif79] is to determine the complexity of deciding whether a player's probability of winning with the optimal strategy is exactly 1/2.

There are many possible modifications to this problem. For example: Deciding whether the probability of winning is greater than 1/2; Deciding whether the probability of not losing is smaller than 1/2 (a player does not lose if he either wins, or the game does not end); Approximating the winning probability to within a fixed error; Finding the best move in a given position; etc. The following corollary (with slight modifications) is robust enough to hold with respect to any of the above definitions.

Corollary 2.4.19 *The question whether $player_1$ has a probabilistic strategy which wins with probability $> 1/2$ in a reasonable game of imperfect information is undecidable.*

Proof: As we saw in Theorem 2.3.15, for any r.e. language L one can build a system where V accepts x ($player_1$, claiming $x \in L$, wins with high probability) if and only if $x \in L$. Thus determining whether $player_1$ has a strategy which is expected to win is equivalent to determining whether $x \in L$. ◇

2.5 Simultaneous space and time bounds

In section 2.3 we proved that a constant space (pfa) verifier can simulate a Turing machine. In this section we show how a constant space verifier can simulate another verifier which is not space bounded. Doing this may seem pointless, as a constant space verifier is as strong as a Turing machine, and an unbounded space verifier cannot be stronger. But the point is that when a Turing machine simulates an interactive proof, there is an exponential overhead in time, and so simulating an unbounded space verifier by a constant space verifier via a Turing machine is expensive. In this section we describe a direct simulation which has only polynomial overhead. This will allow us to transform results concerning poly-time verifiers to systems where the verifier has simultaneous poly-time and constant space bounds.

Consider a poly-time verifier V_{poly}. It has an input tape, a polynomial size work tape, access to random bits, and access to two oracles. If we want to simulate V_{poly}'s computation using V which has only constant space, all we have to do is simulate the work tape. All the rest (input tape, random bits and two oracles) V already has. In Theorem 2.3.14, we saw how a work tape can be simulated using two oracles. This simulation has only polynomial overhead (at most the square of the computation time). The problem is that in this simulation the oracles are aware of the contents of the work tape being simulated, and so they become aware of V_{poly}'s state. This implies that this simulation works only in cases where the original protocol run by V_{poly} makes no use of secret randomness.

Corollary 2.5.20 *In $IP(\times)$, a pfa verifier can test membership in any P-space language,* *and do so in polynomial time.*

Proof: In [FST88] a protocol is given by which a poly-time verifier can test any P-space assertion. In this protocol all communication is public, and V need not toss coins. By the discussion above, this protocol can be simulated by a constant space verifier in polynomial time. ◇

A natural parallel to Corollary 2.5.20 is that in $IP(\|)$, a pfa verifier can verify in polynomial time any language in $IP(|)$. This can be proved through the equivalence of public coins and private coins for $IP(|)$ [GS86]. But in fact, this result can be strengthened:

Theorem 2.5.21 *In $IP(\|)$, a constant space verifier can verify in polynomial time any language that an unbounded space verifier can accept in polynomial time.*

Remark: In case an oracle is cheating, V rejects with high probability in polynomial time, but there is some small probability that V does not stop in polynomial time (or that V accepts incorrectly).

Proof: In [BGKW88] it is shown that a poly-time verifier with two provers can simulate a system with any number of provers. It is important to note that this simulation can be done by a *synchronous* verifier. That is, the exact points in time at which the verifier sends messages to the two provers are fixed before the protocol begins, and do not depend on contents of messages from the provers, or on the verifier's secret coin tosses. So in order to prove Theorem 2.5.21, it remains to prove the following Lemma:

Lemma 2.5.22 *Both in the $IP(\|)$ model, and in the $IP(\times)$ model, a pfa verifier can verify in polynomial time any statement that a synchronous poly-time verifier (which has no space limitations) can.*

Proof: Consider a verification system with synchronous poly-time verifier V_1. We first construct a new verifier V_2 which follows the same protocol as V_1, but has only one work tape (see [HU79]). Furthermore, with each access to the work tape V_2 makes, we extend the nonblank portion of the tape one cell to the right, either as a natural consequence of V_2's computation, or artificially with a "pseudo-blank" character. This is done in order to ensure that no information about V_2's computation (other than the number of computation steps made) can be derived by knowing the length of the nonblank portion of V_2's work tape. Next, we break each cell in V_2's work tape in two: An odd location and an even location. The value stored in the odd location is chosen at random, with uniform distribution over the values 0 to $q-1$, where each random choice is made independently. The value stored in the even location is computed so that the sum modulo q of the odd and even location gives the correct value of the cell. Thus, V_2 can still use his work tape, but anyone observing either only odd locations or only even locations on the work tape, can learn nothing about V_2's computation. Now we replace V_2 by the pfa V, and use the construction of Theorem 2.3.14 in order to simulate V_2's work tape. By this construction one oracle holds the contents of even locations on the work tape, and the other oracle holds the contents of the odd locations. V simulates V_2's part in the protocol, and whenever V_2's protocol calls for addressing one of the oracles, V does so. The construction guarantees that the oracles cannot gain information about V_2's state in order to send V messages different from what they would send V_2. (This is where we used the fact that V_1 is synchronous. The fact that the oracles can count how many computation steps V makes up to the time they have to send a message does not teach them anything they did not know beforehand.) Thus V simulates V_1, except when cheating is detected (in which case V rejects the cheating oracle). $\diamond$

Using Lemma 2.5.22 and the discussion preceding it, this completes the proof of Theorem 2.5.21. $\diamond$

A recent result of Babai, Fortnow and Lund [BFL90] shows that nondeterministic exponential time languages can be verified by a polynomial time verifier in the collaborating provers model. Combining this with Theorem 2.5.21 we obtain:

Corollary 2.5.23 *In $IP(\|)$, a constant space verifier can verify in polynomial time any nondeterministic exponential time language.*

Chapter 3

Expected Polynomial Time Verifiers

3.1 Introduction

The concept of zero knowledge (ZK) [GMR85] is appealing very much because of its paradoxical philosophical interpretation. The verifier receiving a ZK proof has no idea why the statement being proven is correct, yet he is convinced "beyond any reasonable doubt" that the statement is indeed correct. As is well established, zero knowledge protocols are central building blocks in many cryptographic applications ([GMW86], [FFS87], [GMW87], and others).

The formal definition of zero knowledge involves a simulator which works in *expected* polynomial time, rather than strictly polynomial time. In this chapter we study the effect of this definition. We argue that the verifier in these systems should also be modeled as an expected polynomial time machine. We then show that for this class of verifiers, the proposed simulation of all the known *computational* zero knowledge protocols (e.g., [GMW86], [Blum86]) cannot be carried out even in expected polynomial time.

Related work: There is strong and justified criticism raised by Levin [L86] and explained by Goldreich [G88] on the way "average" computer scientists define the concept of expected polynomial time. Levin suggests an alternative definition for the concept *polynomial on the average*. Our claims are independent of the way expected polynomial time computation is defined, and they continue to hold even if the simulator runs in time *polynomial on the average*, as in Levin's [L86] definition.

3.2 Why Verifiers Should Run in Expected Polynomial Time

In Definition 1.4.4 of zero knowledge, the simulator M is *expected polynomial time.* As remarked in [GMR85], this appears to be necessary for some zero knowledge proof systems. (We discuss this issue in Section 4 of this chapter.) Based on this, we make some philosophical claims:

Claim 3.2.24 *The simulator M should do only things that V' can do by himself.*

Justification: This is the essence of the idea of zero knowledge. V' does not learn anything from P, because he himself can simulate the whole protocol.

Claim 3.2.25 *M must not be of a complexity class higher than the complexity class V' is in.*

Justification: Otherwise, M does things that V' cannot do. If M does not use his extra power, then there is no point of having him in a higher complexity class in the first place.

Claim 3.2.26 *V' must be allowed to run for expected polynomial time (and not just worst case polynomial time).*

Justification: The simulator in zero knowledge protocols is allowed to run for expected polynomial time (see Definition 1.4.4). Thus Claim 3.2.26 follows from Claim 3.2.25.

We demonstrate Claim 3.2.26 by an example. Let f be a one way permutation on $\{0,1\}^n$. Consider the following two party protocol (A, B), in which A is infinitely powerful and B is probabilistic polynomial time:

Protocol 3.2.27 *A chooses randomly with uniform distribution an argument $x \in \{0,1\}^n$. A then chooses randomly with uniform distribution x' satisfying $f(x') \geq f(x)$. A sends (x, x') to B.*

We prove that the above protocol is zero knowledge by exhibiting an expected polynomial time simulation procedure for it:

1. Choose x at random.
2. Repeatedly choose at random new x', until $f(x') \geq f(x)$.

The expected running time of the above simulation is

$$\sum_{i=1}^{2^n} 2^{-n}(\frac{i}{2^n})^{-1} = \sum_{i=1}^{2^n} \frac{1}{i} = O(n).$$

Thus according to Definition 1.4.4, Protocol 3.2.27 is perfect zero knowledge. But consider a strictly polynomial time B. Could he really simulate this protocol by himself? If B could invert the oneway permutation f, then he could pick x at random, then pick $f(x') \geq f(x)$ and invert it to obtain x', thus simulating the protocol. But can B simulate the protocol without breaking the oneway permutation?

In order to discuss this question we model the oneway permutation as a random permutation given in the form of a blackbox. In order to compute f in the forward direction, B may feed the blackbox with the argument x and obtain the output $f(x)$. But given an image y, B has no way of finding its preimage other than by trying sufficiently many possible inputs to the blackbox until it outputs y.

Proposition 3.2.28 *In the blackbox oneway permutation model, Protocol 3.2.27 cannot be simulated in polynomial time.*

Proof: Assume to the contrary that the protocol can be simulated in time n^c by some simulator M', where c is some positive constant. Consider a distinguisher D which outputs 1 iff its input (x, x') satisfies $f(x') > f(x) \geq 2^n(1 - n^{-3c})$. It follows that on transcripts of the real protocol, $\text{Prob}(D \text{ outputs } 1) = n^{-3c}$.

Since f (the blackbox) is a random permutation, then for any argument x, $prob(f(x) \geq 2^n(1 - n^{-3c})) \leq n^{-3c}$. Due to its running time, M' obtains from the blackbox the values of f on at most n^c arguments. Consequently, the probability M' obtains a pair of arguments on which D outputs 1 is $O(n^{-4c})$, and thus D distinguishes between the distribution that M' generates and the executions of the real protocol. $\diamond$

Throughout this thesis, we assume that expected polynomial time algorithms can generate distributions which are computationally distinguishable from any distribution that strictly polynomial time algorithms can generate. This assumption is also implicit in Definition 1.4.4 of zero knowledge, as otherwise there would be no reason to let the simulator run for expected polynomial time. However, the situation is different when we consider the task of distinguishing between distributions.

Proposition 3.2.29 *Any two ensembles which are computationally distinguishable by expected polynomial time algorithms are also computationally distinguishable by strictly polynomial time algorithms.*

Proof: Let $A(x)$ and $B(x)$ be any two ensembles and assume that there exists a distinguisher D for which:

$$|prob(D(A(x)) = 1) - prob(D(B(x)) = 1)| > n^{-k}$$

where D runs in expected time $t(n^c)$, and both c and k are positive constants. By the bound on D's expected running time it follows that the probability it runs for more than $2n^{c+k}$ steps is at most $\frac{n^{-k}}{2}$. Thus even if D is stopped and forced to output 0 whenever it exceeds $2n^{c+k}$ steps, it still satisfies:

$$|prob(D(A(x)) = 1) - prob(D(B(x)) = 1)| > \frac{n^{-k}}{2}$$

$\diamond$

Remark: Let D_r denote the nonuniform polynomial time distinguisher obtained by fixing $D's$ coin tosses to be the random string r (of length $2n^{c+k}$). By a standard averaging argument, for some r:

$$|prob(D_r(A(x)) = 1) - prob(D_r(B(x)) = 1)| > \frac{n^{-k}}{2}$$

Consequently, any two ensembles which are distinguishable by expected polynomial time distinguishers, are also distinguishable by polynomial time deterministic nonuniform distinguishers.

3.3 Why Verifiers Cannot Run in Expected Polynomial Time

To continue our argument, we pause to inspect the nature of the simulator M. In virtually all zero knowledge protocols, the proof that the protocol is zero knowledge involves *blackbox simulation* [Oren87]. In this case, we have one simulator M which can be used for any V', and this M uses V' as a blackbox subroutine. On common input x and auxiliary input y, M supplies the blackbox V' with x, y, and random bits upon demand.

Now we reach the heart of our argument. We show that for *computationally* zero knowledge protocols, there exist a verifier V', such that the following two conditions hold:

1. When interacting with the real prover P, the interactive proof (P, V') takes expected polynomial time.

2. The standard blackbox simulation $M(V')$ does not end in expected polynomial time.

We do not prove this statement for all computational zero knowledge protocols and for all possible simulators $M(V')$. We only demonstrate this phenomenon on one simplified example: Protocol 1.6.13 for proving that an n node graph G is Hamiltonian. However, the same argument applies equally well to all the proposed simulations of published computationally zero knowledge protocols.

We recall that in the simulation of Protocol 1.6.13, M randomly guesses V''s future reply, $'0'$ or $'1'$. If M guesses $'0'$ it commits to a Hamiltonian cycle as P does. But if M guesses $'1'$, it commits to the n node empty graph, something that the truthful prover never does.

For any x, let $E(x)$ be a distribution on strings of length $|x|$. Assume that the family $\{E(x)\}$ of random variables can be generated in expected polynomial time, but cannot even be approximated in polynomial time. Consider now the following cheating V'. V' always replies '1' in move 2 of Protocol 1.6.13, forcing the simulator always to commit to the n node empty graph. When the protocol ends, V' looks at the transcript of the protocol and tries to guess an NP witness which proves that he was interacting with a simulator (such a witness is composed of valid opening of all the committed bits that V' received in move 1, which proves that the graph committed to was empty). Because the commitment scheme is secure, V' has only negligible probability of exposing M, denoted by $\nu(n)$. If V' succeeds, he generates infinitely many random bits and never stops. If V' fails, he outputs a random element $r \in E(x)$ which he generates in expected polynomial time.

When V' interacts with the real prover, the whole protocol takes expected polynomial time, because P never encrypts the n node empty graph. But when V' interacts with the simulator, then if the simulator is exposed, M is required to supply V' with infinitely many random bits. The expected running time of the system $M(V')$ is thus

$$poly(n) + \nu(n)\infty = \infty > poly(n).$$

We note that two simple attempts to patch the simulation process in order to make it expected polynomial time do not work.

1. M cannot restrict V' to some prespecified polynomial number of random bits, because as far as M knows, V' may be trying to generate an element in $E(x)$, and this process cannot be approximated in strictly polynomial time.

2. M cannot supply an element of $E(x)$ by itself, because this would not be blackbox simulation. M which receives the blackbox V' does not know what the distribution $E(x)$ is, and so it has no choice but to use V' in order to generate elements of $E(x)$.

We are thus led to conclude:

Proposition 3.3.30 *There exist cheating expected polynomial time verifiers for which all the known simulation procedures of computational zero knowledge protocols do not end in expected polynomial time.*

3.4 Conclusions

One may counter Proposition 3.3.30 by the following argument: The event that V' exposes M has negligible probability. Hence the problem presented has no real practical significance. We wholly agree with this argument, but point out that similar impractical paradoxes have played a significant role in the foundations of mathematics. In this section we discuss possible ways of answering the claims of the previous sections.

First, it is important to understand exactly why Definition 1.4.4 allows for expected polynomial time simulation. The first definitions of the zero knowledge concept, as presented by Goldwasser, Micali and Rackoff in 1985, required the simulator to be probabilistic polynomial time. Early attempts to prove that certain protocols are zero knowledge met technical difficulties which were solved by allowing expected polynomial time simulation. Subsequently, slight modifications to some of these protocols resulted in construction of polynomial time simulation for them. Still, we can identify two major classes of protocols which seem to require expected polynomial time simulation: Perfect zero knowledge protocols, and constant rounds zero knowledge protocols.

The interactive proof system for the Discrete Log (see Chapter 1) is a good example where expected polynomial time simulation is used to prove the *perfect zero knowledge* property: If some strict polynomial time bound is set on the simulation procedure, then with some negligible but nonzero probability the simulation does not end, and thus the resulting simulated distribution is statistically indistinguishable (rather than perfectly indistinguishable) from the one obtained in real executions of the protocol. For the Discrete Log Protocol, the arguments leading to Proposition 3.3.30 do not apply, and we can allow both simulator and verifier to be expected polynomial time and still preserve the perfect zero knowledge property. Indeed, this seems to be the general case for known perfect zero knowledge protocols. Thus the most problematic protocols are the constant rounds computationally zero knowledge protocols (e.g., see Chapter 8).

One should note that also Definition 1.4.6 of *proofs of knowledge* involves a knowledge extractor which runs in *expected polynomial time*, whereas the prover is only allowed polynomial time computations. Again, expected polynomial time knowledge extraction is required in constant rounds proofs of knowledge (e.g., see Chapter 4).

To give up the concept of expected polynomial time computation altogether would imply giving up constant round computational zero knowledge interactive proofs. We feel that this is a price too heavy to be paid. Thus we present some other alternatives:

1. Find a way of simulating proposed constant round zero knowledge protocols in polynomial time. If this succeeds, there seems to be no reason to allow for expected polynomial time simulation.

2. Retain expected polynomial time simulation, but improve the simulation process. Devise a new simulation procedure for computationally zero knowledge protocols which ends in expected polynomial time even if the verifier is expected polynomial time. Note that we require the simulation to work with any possible cheating verifier, and not just with the particular V' we presented.

3. Define expected polynomial time interactive Turing machines as those which run for expected polynomial time with *any* partner. In this case, V' must run for expected polynomial time with M as well, and Proposition 3.3.30 does not hold. We find this solution unsatisfactory when considering cryptographic applications of the zero knowledge concept. It is unacceptable to ignore a cheating verifier, which manages to extract in polynomial time valuable information from zero knowledge protocols, only because in some other imaginary cases which never occur in practice this verifier is not expected polynomial time.

4. Introduce a new complexity class. One may try to characterize a complexity class which is somewhat higher than expected polynomial time, in which both V' and M can reside with no contradiction. Other than complicating matters, this solution still leaves open the question of what can be said about verifiers which belong to existing complexity classes.

5. Change the interpretation of zero knowledge. The problems arise not because of a mathematical flaw in the combination of polynomial time verifiers and expected polynomial time simulators, but because this seems to violate the intended philosophical

interpretation of the zero knowledge concept. A solution would be to retain the mathematical definition, but to give it a less ambitious interpretation. One such interpretation can be that if polynomial time V' can use P to do some difficult computation, the same computation is *expected* to take V' polynomial time even without the help of P.

One should check that such an interpretation is strong enough for all the intended uses of the zero knowledge concept.

The last alternative is the one we follow throughout this thesis. Namely, verifiers are restricted to probabilistic polynomial time, whereas simulators and knowledge extractors may run for expected polynomial time.

Chapter 4

Compositionallity and Zero Knowledge

4.1 The Case Where Intuition Fails

Consider the two protocols 1.6.12 and 1.6.13 used throughout this thesis. Each one of them has a basic step which is composed of three moves: The prover randomly re-encodes the input problem, the verifier requests either a witness to the fact that the re-encoding was carried out correctly, or a solution to the new problem, and the prover satisfies this request. This procedure is repeated sequentially n times in order to make the probability of successful cheating exponentially small.

One may attempt to reduce the communication costs of this scheme by parallelization: The prover sends all the random encodings simultaneously, the verifier sends all his requests, and the prover satisfies all the requests. Intuitively, it seems as if the parallel version is a zero knowledge proof of knowledge just as the sequential version. It is hard to imagine how the prover may cheat in such a protocol and prove false statements, or how the verifier may attempt to extract any meaningful additional information from the prover. And indeed, this intuition is correct as long as the verifier is honest and chooses his requests independently of P's messages.

Proposition 4.1.31 *The parallel version of Protocols 1.6.12 and 1.6.13 is a zero knowledge proof of knowledge with respect to an honest verifier.*

Proof: The *completeness* property of the protocols follows from the completeness of the basic step of each protocol.

In order to prove the *soundness* property, we construct an *expected polynomial time* knowledge extractor M. As in the sequential version of the protocols, if M obtains two executions of the protocol in which the prover satisfies two different requests on the same randomized instance, then he can compute a witness to the input instance.

M begins by simulating the system (P, V), treating P as a blackbox and truthfully performing V's part of the protocol by himself. If V does not accept P's proof, M stops. Otherwise, M repeatedly simulates the system (P, V), with a fresh random string for V but without changing P's random string, until a second successful execution is achieved. M outputs the witness derived from the two successful executions (unless in both executions V happened to use exactly the same random string) and stops.

Let q_{r_P} denote the probability (taken over V's coin tosses) that V accepts P's proof when P uses the random string r_P. Then the expected running time of the knowledge extraction procedure is

$$(1 + q_{r_P}\frac{1}{q_{r_P}})(time(P) + time(V)) = 2(time(P) + time(V))$$

which is polynomial and independent of the particular random string that P is using.

M fails only if in both successful executions V happened to send exactly the same set of requests. Since $q_{r_P}2^n$ is the number of random strings on which V accepts, the probability of the above event is $\frac{q_{r_P}}{q_{r_P}2^n} = 2^{-n}$, which is negligible.

The following simulation procedure proves that the protocol is *zero knowledge* with respect to V which selects his requests independently of P's messages.

1. Randomly re-encode the input instance using exactly the same procedure that the real prover is using. (This does not require knowledge of a witness.)

2. Receive V's requests.

3. Replace the messages sent at move 1 by new random messages for which V's requests can be met (the same procedure that is used in simulating the sequential version of the protocols). Satisfy these requests.

Since V selects his requests independently of the messages that he receives, the distribution of V's requests in the simulated view is exactly the same as the distribution of V's requests in real executions of the protocol. For Protocol 1.6.12, this implies that the simulated view is perfectly indistinguishable from the real execution. For Protocol 1.6.13, in which re-encoding of the input instance is based upon bit commitment schemes, the simulated view is computationally indistinguishable from the real execution. $\diamond$

The question of whether the parallel version is zero knowledge even with respect to a cheating verifier has received much attention. It was soon realized that known simulation procedures fail in this case, since a cheating V may choose his requests as a wild hashed value of P's first message. Goldreich and Krawczyk [GKr] made this intuition formal by proving that no three move interactive proof is "blackbox simulation" zero knowledge, unless the statement being proved is in BPP. However, it was not known whether cheating V can somehow extract useful information from the prover, or whether the difficulties in demonstrating that the parallel version reveals no additional information are merely technical. In fact, there was at least one known example (the parallel version of the Fiat-Shamir identification protocol) where a three move proof of knowledge for a statement not known to be in BPP provably does not reveal the prover's witness [FFS87].

An important contribution of subsequent chapters of this thesis is in the characterization of cases in which parallelization of zero knowledge protocols does not reveal the prover's witness. But first, in the next section we show that there is a real danger in parallelization of zero knowledge proof.

4.2 Playing Against Two Grandmasters

In this section we demonstrate that P's secret witness may be disclosed if a zero knowledge protocol is composed twice with itself. A related result was independently obtained by Goldreich and Krawczyk [GKr], who prove that "Computational Zero-Knowledge is not closed under parallel composition". But there is a fundamental difference between the two results. Goldreich and Krawczyk state their Theorem in the model where the prover is infinitely powerful, and they use this in an essential way in the proof. In contrast, our result applies to the cryptographically interesting case in which the prover is polynomial time with auxiliary input. Furthermore, [GKr]'s proof applies only to computationally zero knowledge protocols, whereas our proof applies also to perfect zero knowledge protocols. On the other hand, our result relies on intractability assumptions (which is unavoidable as the prover is polynomial time), while [GKr] does not use unproven assumptions.

Theorem 4.2.32 *There exists a zero knowledge proof of knowledge system $(\bar{P}, \bar{V})$ for the discrete log, which when executed twice in parallel discloses the discrete log of the input.*

Proof: Let (P, V) be any zero knowledge proof of knowledge system for the discrete log problem (e.g., Protocol 1.6.12). We construct ($\bar{\text{P}}$,$\bar{\text{V}}$) directly from (P, V).

1. On input (p, g, c, x), $\bar{\text{V}}$ tries to randomly guess w, the unique discrete log of x, satisfying $g^w = x \bmod p$. If $\bar{\text{V}}$ succeeds (with negligible probability), he sends 1. Otherwise he sends 0.

2. If $\bar{\text{V}}$ sent 1 in move 1, he now proves to $\bar{\text{P}}$ in zero knowledge that he knows w, using the protocol (P, V) with reversed roles. If $\bar{\text{P}}$ is convinced by $\bar{\text{V}}$'s proof (this is expected to happen with overwhelming probability with truthful $\bar{\text{P}}$ and $\bar{\text{V}}$), he sends w to $\bar{\text{V}}$, showing that he too knows w, and $\bar{\text{V}}$ accepts. If $\bar{\text{P}}$ is not convinced by $\bar{\text{V}}$'s proof, $\bar{\text{P}}$ stops and $\bar{\text{V}}$ rejects.

3. If $\bar{\text{V}}$ sent 0 in move 1, $\bar{\text{P}}$ proves his knowledge of w using the standard proof system (P, V).

Lemma 4.2.33 *The protocol ($\bar{P}, \bar{V}$) is a complete, sound and perfect zero knowledge proof of knowledge.*

Proof: The properties of ($\bar{\text{P}}$, $\bar{\text{V}}$) follow from the corresponding properties of (P, V).

Completeness: If in move 1 $\bar{\text{V}}$ succeeds in guessing w, then he can perform the role of P in the interactive proof of knowledge (P, V). From the completeness of (P, V) it follows that truthful $\bar{\text{P}}$, who plays the role of V, will accept this subprotocol. Consequently $\bar{\text{P}}$, who really knows w, sends w to $\bar{\text{V}}$, and $\bar{\text{V}}$ accepts.

If in move 1 $\bar{\text{V}}$ fails to guess w, the completeness property of ($\bar{\text{P}}$, $\bar{\text{V}}$) follows directly from that of (P, V).

Soundness: If in move 1 $\bar{\text{V}}$ fails to guess w, the soundness property of ($\bar{\text{P}}$, $\bar{\text{V}}$) follows directly from that of (P, V).

If in move 1 $\bar{\text{V}}$ does guess w, the soundness of the protocol may be argued on many different levels, each of them sufficient for our purpose:

1. The event that $\bar{\text{V}}$ guesses w has negligible probability, and thus can be dismissed.

2. If $\bar{\text{V}}$ guesses w, then certainly he can exhibit w as required from the knowledge extraction procedure.

3. In order for $\bar{\text{V}}$ to accept, even a cheating $\bar{\text{P}}$ eventually must send w, and this w can be output by the knowledge extractor.

Remark: We can also prove that if $\bar{\mathrm{V}}$ sends 1 in move 1, then $\bar{\mathrm{P}}$ must actually know w at this stage (and not learn w from $\bar{\mathrm{V}}$ during he protocol) in order to have nonnegligible probability of convincing $\bar{\mathrm{V}}$ to accept. We construct a knowledge extractor M which extracts w from $\bar{\mathrm{P}}$ without using $\bar{\mathrm{V}}$'s knowledge of w. M uses the simulator guaranteed to exist for the zero knowledge protocol (P, V) in order to reconstruct $\bar{\mathrm{P}}$'s view after he performs V's part in (P, V). Now if $\bar{\mathrm{P}}$ outputs w then obviously he could not have learned w from $\bar{\mathrm{V}}$.

Zero knowledge: We exhibit a simulator M which for any (possibly cheating) verifier $\bar{\mathrm{V}}$, produces in expected polynomial time a view which is perfectly indistinguishable from $\bar{\mathrm{V}}$'s view in the protocol ($\bar{\mathrm{P}}$, $\bar{\mathrm{V}}$). M simulates the role of $\bar{\mathrm{P}}$, and uses $\bar{\mathrm{V}}$ as a blackbox. If in move 1 $\bar{\mathrm{V}}$ sends 1, M has to perform V's role in the protocol (P, V), with $\bar{\mathrm{V}}$ acting as the prover P. If M sees that V rejects (P, V), M ends the simulation as the real $\bar{\mathrm{P}}$ would stop the protocol. If M sees that V accepts (P, V), M uses the knowledge extractor which is guaranteed to exist for the proof of knowledge (P, V) and extracts in expected polynomial time w from $\bar{\mathrm{V}}$. Then M completes the simulation by outputting w.

If V sends 0 at move 1, then the simulator for (P, V) can be used directly to prove the zero knowledge property of ($\bar{\mathrm{P}}$, $\bar{\mathrm{V}}$). $\diamond$

Consider now two executions, $(\bar{P}_1, \bar{V})$ and $(\bar{P}_2, \bar{V})$ in parallel. A cheating verifier V can always extract w from $\bar{P}_1$ and $\bar{P}_2$ by using the following strategy: In move 1, V sends 0 to $\bar{P}_1$ and 1 to $\bar{P}_2$. Now V has to execute the protocol (P, V) twice: Once as a verifier talking to the prover $\bar{P}_1$, and once as a prover talking to the verifier $\bar{P}_2$. This he does by serving as an intermediary between $\bar{P}_1$ and $\bar{P}_2$, sending $\bar{P}_1$'s messages to $\bar{P}_2$, and $\bar{P}_2$'s messages to $\bar{P}_1$. Now $\bar{P}_2$ willfully sends w to V. $\diamond$

Remark 1: Assuming the intractability of the discrete log, Theorem 4.2.32 proves that zero knowledge is not preserved under parallel composition.

Remark 2: We emphasize the importance of the fact that x has a *unique* witness w. Otherwise a single execution of the protocol ($\bar{\mathrm{P}}$,$\bar{\mathrm{V}}$) would not be zero knowledge, as a verifier which knows one of the witnesses for x (e.g., has it as auxiliary input) would learn which witness $\bar{\mathrm{P}}$ has as auxiliary input. This fact cannot be deduced by a simulator M just by observing x and $\bar{\mathrm{V}}$.

Remark 3: Theorem 4.2.32 generalizes to any other relation $R = \{(x, w)\}$ which has a zero knowledge interactive proof of knowledge system, provided each instance has a unique witness.

Chapter 5

Witness Indistinguishable Protocols

5.1 Introduction

Zero knowledge protocols are very useful primitives in the design of cryptographic schemes (see [GMW86], [GMW87]). But as we have seen in Chapter 4, the zero knowledge property is not preserved under general composition of protocols, and in particular parallel composition. This is a serious drawback for several reasons:

1. A natural complexity measure for cryptographic protocols is the number of rounds of communication that they require. In order to minimize this measure it is desirable to send messages in parallel, rather than sequentially. If zero knowledge subprotocols are involved, parallelization may jeopardize the security of the system.
2. In order to simplify the design of complex protocols, it is convenient to design them as composition of simpler subprotocols. But even if each of the subprotocols is zero knowledge, it may be the case that their composition is not.
3. If several protocols are run concurrently in an asynchronous fashion, a cheating party might use information from the execution of some of the protocols in which it is participating in order to compute its responses in other protocols, thus effectively composing these protocols, contrary to the intentions of the designer of the system.

Our goal in the rest of this thesis is to underline general principles which can be used to solve the problem of compositionally of cryptographic protocols, and to show how these principles can be applied to overcome the difficulties associated with some cryptographic schemes based on zero knowledge protocols. We introduce two new concepts, *witness indistinguishability* and *witness hiding.* Whereas zero knowledge means that no additional information whatsoever leaks from the prover to the verifier in a cryptographic protocol, the new concepts only limit the transfer of specific types of information (e.g., the prover's secret witness). Nevertheless, these concepts suffice in many cryptographic applications where the concept of zero knowledge may be an overkill.

In this chapter we introduced the concept of *witness indistinguishability.* Informally, a two party protocol in which party P uses one of several secret witnesses to an NP assertion is *witness indistinguishable* if party V cannot tell which witness P is actually using (even when these witnesses are given to V as auxiliary knowledge). We prove that, unlike zero knowledge protocols, witness indistinguishability is preserved under arbitrary composition of protocols, including parallel execution. Thus one may plug a WI protocol into any cryptographic scheme and be sure that the protocol retains its qualities.

An intuitive explanation of the usefulness of witness indistinguishable protocols is offered through the more natural concept of witness hiding. Informally, a protocol is *witness hiding* if by the end of the protocol party V cannot compute any new witness which he did not know before the protocol began. In Chapter 7 we prove that if a statement has at least two independent witnesses, then any witness indistinguishable protocol for this statement is also witness hiding.

The applications which we suggest for witness indistinguishable and witness hiding protocols are applications where compositionally is required. These include giving polynomially many independent noninteractive zero knowledge proofs using the same common random reference string (Chapter 6), construction of identification schemes (Chapter 7), and construction of constant round zero knowledge arguments (Chapter 8) (the relevant terminology for each application is explained in the respective chapter).

5.2 Witness Indistinguishability

We consider two party protocols in which both P (prover) and V (verifier) are polynomial time. P and V see a common input x, and P has a secret auxiliary input: a *witness* w from the witness set $w(x)$. P's purpose is to perform a computational task that would be difficult to perform had P not known w. Typical examples are to give an interactive proof that he "knows" a witness to the NP statement x (in this case w may be the NP witness), or to digitally sign messages which can later be checked by the public key x (in this case w is P's private key). Informally, these protocols are *witness indistinguishable* if V cannot distinguish between two executions of the protocol which differ in the specific witness P is using (even if V knows both witnesses to the statement being proved). For example, let x be a Hamiltonian graph with several Hamiltonian cycles. A proof of Hamiltonicity is WI if V's view is the same no matter which cycle w in x party P knows and uses.

Definition 5.2.34 *Proof system (P,V) is perfectly (statistically) (computationally)* witness indistinguishable *(WI) over R if for any V', any large enough input x, any $w_1, w_2 \in w(x)$, and for any auxiliary input y for V', the ensembles, $V'_{P(x,w_1)}(x,y)$ and $V'_{P(x,w_2)}(x,y)$, generated as V's view of the protocol, are perfectly (statistically) (computationally) indistinguishable.*$\diamond$

Remark: Unlike definition 1.4.7 (for ZK), the definition for WI involves no simulator M.

The next Theorem shows a simple relation between ZK and WI protocols.

Theorem 5.2.35 *Let (P,V) be any zero knowledge protocol. Then the protocol is witness indistinguishable.*

Proof: The proof follows from the fact that simulation of zero knowledge proofs of knowledge is independent of the particular witness the prover may be using. Consider any protocol (P,V) which is witness distinguishable. That is, there exists a nonuniform verifier V', a distinguisher D, a constant c, and infinitely many triples (x, w_1, w_2) (where $w_1, w_2 \in w(x)$), such that:

$$|Prob(D(V'_{P(x,w_1)}) = 1) - Prob(D(V'_{P(x,w_2)}) = 1)| > 1/n^c$$

Protocol (P,V) cannot be zero knowledge. For any proposed simulator M (even one that performs exponential time computations!), $Prob(D(M(x) = 1)$ differs from either $Prob(D(V'_{P(x,w_1)}) = 1)$ or $Prob(D(V'_{P(x,w_2)}) = 1)$ by at least $1/2n^c$. Since both the latter cases are valid distributions of interactive proofs for x, we conclude that D is a distinguisher which fails any simulator.$\diamond$

5.3 Compositionality

Imagine a cryptographic community with several types of protocols: Identification protocols, key exchange protocols, etc. Each type of protocol may be executed many times, with different inputs and different participating parties. Each party may take part in several protocols, and execute these protocols with any sort of interleaving between their steps.

Though it may not be the intention of the designer of such a system to compose together several protocols, this situation may be created by a (cheating) party which uses information from the execution of some protocols in which it is participating in order to compute its responses in other protocols.

In order to discuss composition of protocols, we extend the notion of an *interactive Turing machine*, and allow it to have several *communication ports*, each one addressing a different communication tape. For convenience, we assume that the ports are arranged in some linear order. At each unit of time, the interactive Turing machine is connected to one of the ports, and acts as an ordinary interactive Turing machine with respect to the communication tape it can address. However, in a single computation step, the machine may decide (based on the state of its finite control and current input characters) to shift one port to the right or to the left, enabling it to access other communication tapes.

In order to discuss the effect of these arbitrary compositions on the WI property, we consider the asymptotic behavior of whole *systems* of protocols, as the parameters of the system grow. Thus, if n is a parameter denoting size, then the number of parties, their running time, the sizes of their descriptions, and the sizes of individual inputs to each protocol are all bounded by some polynomial in n. On the other hand, the types of protocols which can be run in such a system are fixed in advance, and their number does not grow with n.

Definition 5.3.36 *For some constant t, let R^j (for $1 \leq j \leq t$) be relations testable in polynomial time, and let (P^j, V^j) be respective proof systems for these relations. The* general composition *of protocols (P_i, V_i), each one of which is from one of the t types mentioned above, is their concurrent execution. The provers and verifiers in the different protocols need not be pairwise distinct, and the same holds for the inputs x_i and the relations R^i. Each participant runs his own polynomial time* scheduling procedure *which governs the order in which he executes elementary steps of the different protocols in which he participates. For convenience we assume that the inputs x_i to the protocols are all of the same size n. The composition is* polynomial *if there is one polynomial in n bounding the number of participants, the sizes of their descriptions, and their running times.*

We single out two special cases of the general composition: *Sequential composition*, in which the protocols are executed one after the other, and *parallel composition*, in which all protocols are of the same type and run on the same input, and for each j, steps j in all the protocols are executed at the same time.

Definition 5.3.37 *In a composition of protocols, party A* faithfully *follows his role as prover in proof system (P, V), if the following two conditions hold:*

1. *In computing his messages as prover in the proof system (P, V), party A uses only the algorithm P, and gives P as inputs only the common input x, the witness $w \in w(x)$ which A has as an auxiliary input, independent coin tosses, and V's messages. P does not use information from any other protocol in which A may be participating. Likewise, A does not use information from the protocol (P, V) (e.g., the coin tosses) in other protocols.*

2. *A's scheduling procedure is independent of the particular witness he has for x.*

Definition 5.3.38 *A polynomial composition of protocols is* witness indistinguishable, *if for any subset of provers $\mathcal{P} = (P_1, P_2, ..., P_k)$ which follow their protocols faithfully, and any two sets of respective witnesses $\mathcal{W}^1 = (w_1^1, w_2^1, ...w_k^1)$ and $\mathcal{W}^2 = (w_1^2, w_2^2, ...w_k^2)$, it is indistinguishable to the coalition of all the other provers and verifiers whether $\mathcal{P}$ are using $\mathcal{W}^1$ or $\mathcal{W}^2$ (for large enough n, where k may be a function of n).*

Remarks:

1. It may happen that the same party A plays a double role: It faithfully executes its role as prover P in one proof system, and in the same time it is put in the coalition of bad provers because of its role in other protocols. If this is the case, then logically split A into two parts: The part implementing P, which behaves truthfully; and the rest of A, which governs the scheduling of P, but does not control, nor view, P's auxiliary input or random coin tosses.

2. It may happen that the coalition of bad participants may know which witness a faithful prover is using in a witness indistinguishable protocol, through some external condition on the whole set of protocols (e.g., the inputs to the proof systems may be chosen in some correlated way). However, this does not contradict the witness indistinguishability property, since it is not the execution of the protocol which reveals the witness to the verifier.

Theorem 5.3.39 *WI is preserved under polynomial composition of protocols.*

Proof: Consider polynomially many protocols carried out concurrently (sequentially, in parallel, or with any type of interleaved steps). Assume that for infinitely many n, $\mathcal{P}(n)$ are subsets of the provers who carry out their WI protocols faithfully and for them WI is not preserved. That is, there exist sets of verifiers $\mathcal{V}(n)$, auxiliary inputs $\mathcal{Y}(n)$ to $\mathcal{V}(n)$, and sets of witnesses $\mathcal{W}^1(n)$ and $\mathcal{W}^2(n)$, such that the two ensembles $\mathcal{V}_{\mathcal{P}(\mathcal{W}^1)}(\mathcal{Y}(n))$ and $\mathcal{V}_{\mathcal{P}(\mathcal{W}^2)}(\mathcal{Y}(n))$ are polynomially distinguishable. By the "hybrid" argument of [GM84], there must be a "polynomial jump" somewhere in the execution: For any n, there exists k, such that if all $P \in \mathcal{P}(n)$ with index less than k use witnesses from $\mathcal{W}^1$, and all $P \in \mathcal{P}(n)$ with index greater than k use witnesses from $\mathcal{W}^2$, the ensembles which differ only in the witness P_k is using are distinguishable by $\mathcal{V}$. Now we use the auxiliary input of the verifier to derive a contradiction. The whole set of protocols which are taking place concurrently can be simulated by a modified V', who has as auxiliary input the algorithms and auxiliary inputs of all other participants (including $\mathcal{Y}(n)$, $\mathcal{W}^1(n)$ and $\mathcal{W}^2(n)$). We use here the fact that the composition is polynomial: there are only polynomially many participants, and each one of them is polynomial time (including the provers). This random polynomial time V' can now distinguish between truthful $P_{k(n)}$ using $w^1_{k(n)}$ or $w^2_{k(n)}$. Since there are only finitely many (t) types of protocols (by Definition 5.3.36), then the WI property is violated on infinitely many inputs for at least one of these types of protocols. This contradicts our assumption that the original protocol was witness indistinguishable. $\diamond$

Remark: The proof of Theorem 5.3.39 uses the fact that the truthful prover need not be stronger than polynomial time. If the real prover is allowed to perform exponential time computations, the theorem is false. This can be proved in a way similar to [GKr]'s proof that zero knowledge is not preserved under parallel composition.

The above Theorems establish a methodology for constructing WI protocols. Take the basic step of a ZKIP. By Theorem 5.2.35 it is also WI. Iterate the basic step n times in parallel. This is probably not zero knowledge [GKr], but by Theorem 5.3.39, it is WI. In particular, we get:

Corollary 5.3.40 *Under the assumption that oneway functions exist, any NP language has a constant round WI proof system.*

Chapter 6

Non-Interactive Zero Knowledge

6.1 Background

Blum, Feldman, and Micali [BFM88] suggested the intriguing concept of *noninteractive zero knowledge* (NIZK), aimed at eliminating the interaction between prover and verifier in zero knowledge interactive proof systems. The prover P writes down a zero knowledge proof that an input x belongs to a prespecified language L, and any verifier V can check the validity of this written proof against a universal publicly available random string (such as the RAND string of one million random digits), called the *reference string.* NIZK has become an important primitive for cryptographic protocols, with applications such as signature schemes [BG89] and encryption schemes secure against chosen ciphertext attack [NY90].

NIZK proof systems for any NP statement were constructed in [BFM88] and [DMP87], under a specific number theoretic assumption. The main disadvantage of these *bounded* NIZK proofs is that the prover can prove only one statement of size bounded by the length of the reference string: If polynomially many proofs are given using the same reference string, the zero knowledge property breaks down[1]. In [BDMP89] it was finally shown how a single prover can give polynomially many proofs using the same reference string, but the scheme is still based on a specific number theoretic assumption, and cannot support polynomially many provers.

In this chapter we suggest a different solution to this problem. We show how to transform any bounded NIZK proof system for an NP complete language into a *general* NIZK proof system in which polynomially many independent provers can share the same reference string

[1] The method suggested in [BFM88] for overcoming this difficulty was found to be flawed.

and use it to prove polynomially many statements of polynomial length. The transformation is based on the general assumption that oneway functions exist. Thus any weakening in the cryptographic assumptions made in the construction of bounded NIZKs is automatically reflected in the cryptographic assumptions made in the construction of general NIZKs. This is of particular importance, since Lapidot and Shamir [LS90] show how to construct bounded NIZK for any NP statement based on the general, rather than number theoretic, assumption that trapdoor permutations exist.

Independently of our work, De Santis and Yung [DY] also show how to transform bounded NIZK proof systems into general ones, though their transformation produces noninteractive proofs which are longer than ours by several orders of magnitude.

We now give a quick overview of our construction. It is based on the concept of witness indistinguishability. We prove that any NIZK proof system in which the prover need not be stronger than polynomial time is also witness indistinguishable. We then prove that the witness indistinguishability property is preserved even if polynomially many noninteractive witness indistinguishable proofs are given using the same common reference string (again, provided that the prover in each individual proof need not be stronger than polynomial time). These two theorems offer a noninteractive parallel to the case of interactive proofs (see Theorems 5.2.35, 5.3.39), with the additional complication of dependencies created by different noninteractive proofs sharing the same common reference string.

The last step in our construction of general NIZKs is to show that any sequence of noninteractive witness indistinguishable proofs is also zero knowledge. But this claim is not always true. To solve this problem we show how, under the assumption that oneway functions exist, one can modify any NIZK proof system for any NP complete language to a new noninteractive proof system for which witness indistinguishability always implies zero knowledge.

6.2 Noninteractive Witness Indistinguishability

Let $R = \{(x, w)\}$ be a relation testable in polynomial time and let L_R be the NP language $\{x \mid \exists w \; s.t. \; (x, w) \in R\}$. In a *noninteractive zero knowledge proof system* for L_R, the prover and verifier share a random reference string which is denoted by σ. The prover's goal is to write down a proof that a common input x of length n belongs to L_R. The verifier may check the validity of this proof against the reference string σ. Throughout this chapter we assume that both P and V are probabilistic polynomial time Turing machines. Consequently, they may use only a polynomial size prefix of σ. As always with polynomial time provers, P has an auxiliary input w, which is the witness to the NP statement $x \in L_R$.

Definition 6.2.41 *A* noninteractive proof system *for an NP language L_R characterized by a relation R is a pair of probabilistic polynomial time algorithms (P, V) satisfying:*

1. Completeness: $\forall (x, w) \in R$

   Prob($V(x, \sigma, P(x, w, \sigma))$ accepts) $> 1 - \nu(n)$

2. Soundness: *Even a cheating prover P' cannot convince V to falsely accept x on a randomly chosen σ. Formally:*

   $\forall P' \, \forall x \notin L_R \, \forall w'$

   Prob($V(x, \sigma, P'(x, w', \sigma))$ accepts) $< \nu(n)$

The probabilities are taken over the choices of σ, and over the coin tosses of P and V.

Remark: A relaxed soundness condition restricts cheating provers to random polynomial time computations, and then the probabilities are taken over the random coin tosses of P' as well. A more severe soundness condition changes the order of quantification and states that for almost any choice of σ, no false statement can be proved. The results of this chapter are independent of the particular soundness condition which is used.

Definition 6.2.42 *A noninteractive proof system for relation R is* bounded zero knowledge *if there exists a random polynomial time simulator M such that for any $(x, w) \in R$, the two ensembles $(x, \sigma, P(x, w, \sigma))$ and $M(x)$ are computationally indistinguishable (by nonuniform polynomial time distinguishers). Formally:*

$\exists M$ s.t. $\forall D \, \forall (x, w) \in R$

$|\text{Prob}(D(M(x)) = 1) - \text{Prob}(D(x, \sigma, P(x, w, \sigma)) = 1)| < \nu(n)$

The probabilities are taken over the choices of σ, and over the coin tosses of P and M.

Using a bounded NIZK proof system the prover can prove one statement in zero knowledge. It is not known whether in general the zero knowledge property is preserved if the same common reference string σ is used to prove more than one statement. This is an obvious drawback when cryptographic applications are concerned.

Definition 6.2.43 *A noninteractive proof systems for the language L_R is* general zero knowledge *if there exists a random polynomial time simulator M such that for any polynomial sequence of instances $(x_1, w_1) \in R$, $(x_2, w_2) \in R$, $(x_3, w_3) \in R$, ... the two ensembles $(\sigma, x_1, P(x_1, w_1, \sigma), x_2, P(x_2, w_2, \sigma), ...)$ and $M(x_1, x_2, ...)$ are polynomially indistinguishable.*

Remarks:

1. An important feature of our definition of general zero knowledge noninteractive proof systems is that each of the statements x_j is proven independently. Consequently, polynomially many provers can share the same random reference string σ and prove polynomially many statements independently. A somewhat weaker definition, in which the proof of statement x_j may depend on the proof of previous statements, is given in [BDMP89]. Their definition applies only to the case that a single prover uses σ to prove polynomially many statements. The construction that they propose does not support polynomially many independent provers.

2. In cryptographic scenarios, one may wish to consider *adaptive* attacks, in which an adversary choses the input instances to the various proof systems, and possibly bases these input instances on the contents of the random reference string, and on transcripts of previously given noninteractive proofs. The techniques of this chapter can also be used to retain the zero knowledge property in adaptive scenarios. See [FLS90] for details.

Our goal in this chapter is to transform bounded NIZK proof systems for an NP complete language L_R into general NIZK proof systems for the same language L_R. As a first step towards this goal, we turn to consider noninteractive witness indistinguishable (NIWI) proofs.

Definition 6.2.44 *Noninteractive proof system* (P, V) *is* bounded witness indistinguishable *over* R *if for any large enough input* x*, any* $w_1, w_2 \in w(x)$*, and for a randomly chosen reference string* σ*, the ensembles which differ only in the witness that* P *is using, but not in* x *or* σ*, are computationally indistinguishable. Formally:*

$\forall D \, \forall x \in L_R \, \forall w_1, w_2 \in w(x)$

$|\text{prob}(D(x, \sigma, P(x, w_1, \sigma)) = 1) - \text{prob}(D(x, \sigma, P(x, w_2, \sigma)) = 1)| < \nu(n)$

The probability space is that of the random choices of σ *together with* P*'s random coin tosses.*

Theorem 6.2.45 *Any bounded noninteractive zero knowledge proof system with polynomial time prover is also a bounded noninteractive witness indistinguishable proof system.*

We note a subtle point: Theorem 6.2.45 is not a special case of Theorem 5.2.35 which proves the implication ZK $\Longrightarrow$ WI for *interactive* proof systems. Consider for example a distinguisher D, which for half of the choices of the public string σ is biased towards noninteractive proofs which use the witness w_1, and for the other half of the choices of σ

is biased towards noninteractive proofs which use the witness w_2. When averaging over all possible choices of σ, D has the same probability p of outputting 1, whichever of the two witnesses is used. Furthermore, it is possible that there exists a simulator M which produces strings on which D has probability p of outputting 1. Thus D may serve as a distinguisher which violates the WI property, without D violating the ZK property.

Proof (Theorem 6.2.45): Assume that for infinitely many triplets (x, w_1, w_2) of inputs together with their respective witnesses, D can distinguish between P using witness w_1 and P using witness w_2. Formally, for some $k > 0$:

$$\sum_{\sigma}(2^{-|\sigma|}|Prob(D(x,\sigma,P(x,w_1,\sigma)) = 1) - Prob(D(x,\sigma,P(x,w_2,\sigma)) = 1)|) \ > n^{-k}$$

where the probabilities are taken over the random choices of P. As a first step of our proof, we remove the absolute values in the above sum, without significantly reducing the quality of the distinguishing process. We do so by constructing a nonuniform *random* polynomial time distinguisher D', which uses knowledge of both w_1 and w_2. (D' can be transformed into a *deterministic* nonuniform distinguisher by standard averaging techniques. See remark following Proposition 3.2.29.) On input z, a noninteractive proof for x using the public random string σ, D' generates n^{k+1} independent strings from each of the distributions $P(x, w_1, \sigma)$ and $P(x, w_2, \sigma)$ (once again, we note that this is possible because P is polynomial time, and D' can simulate P with the relevant auxiliary input). D' feeds these strings to D, and determines by a majority vote whether D is biased towards w_2 on σ. Then D' feeds D with z, and flips the decision made by D if and only if the bias test showed a bias towards w_2.

It is a simple matter to show that:

$$\sum_{\sigma}(2^{-|\sigma|}|Prob(D'(x,\sigma,P(x,w_1,\sigma)) = 1) - Prob(D'(x,\sigma,P(x,w_2,\sigma)) = 1)|) \ > \frac{1}{2n^k}$$

Now we can apply the proof of Theorem 5.2.35 to show that the original protocol was not zero knowledge. $\diamond$

Remark: The above proof uses the fact that P is polynomial time. It does not hold for exponential time provers. In fact, the truthful exponential prover in [LS90]'s protocol is deterministic (the only randomization is in the choice of σ), and so their protocol cannot be WI.

Definition 6.2.46 *A noninteractive proof system is* general witness indistinguishable *over R if for any polynomial sequence of instances with respective pairs of witnesses, the two* *ensembles* $(\sigma, P(x_1, w_1^1, \sigma), P(x_2, w_2^1, \sigma), P(x_2, w_3^1, \sigma), ...)$ *and* $(\sigma, P(x_1, w_1^2, \sigma), P(x_2, w_2^2, \sigma), P(x_2, w_3^2, \sigma), ...)$ *are computationally indistinguishable.*

Theorem 6.2.47 *Any bounded noninteractive witness indistinguishable proof system is also general witness indistinguishable.*

Proof: Assume that for infinitely many n there exist polynomial sets of inputs $\mathcal{X}(n)$ and sets of witnesses $\mathcal{W}^1(n)$ and $\mathcal{W}^2(n)$, such that the two ensembles $P(\mathcal{X}, \mathcal{W}^1, \sigma)$ and $P(\mathcal{X}, \mathcal{W}^2, \sigma)$ are polynomially distinguishable. By the "hybrid" argument of [GM84], there must be a “polynomial jump” somewhere in the execution: For any n, there exists k, such that if for all $x \in \mathcal{X}(n)$ with index less than k the prover uses witnesses from $\mathcal{W}^1$, and for all $x \in \mathcal{X}(n)$ with index greater than k the prover uses witnesses from $\mathcal{W}^2$, then the ensembles which differ only in the witness used for x_k are distinguishable by some distinguisher D. We use the nonuniformity of the distinguishers to derive a contradiction. The whole set of proofs can be simulated by a modified D', who has as auxiliary input the inputs and witnesses to all other proofs. We use here the fact that the prover is polynomial time, and that there are only polynomially many inputs of polynomial length. This random polynomial time D' can now distinguish between proofs for $x_{k(n)}$ in which the prover uses $w_{k(n)}^1$ and proofs in which the prover uses $w_{k(n)}^2$. This contradicts our assumption that the original protocol was witness indistinguishable. $\diamond$

6.3 Multiple NIZK Proofs Based on a Single Random String

We assume the existence of cryptographically secure pseudo-random bit generators [BM84, Yao82], which extend n-bits random seeds to $2n$-bits pseudo-random strings, computationally indistinguishable from strings of truly random $2n$ bits. The existence of pseudo-random generators follows from the assumption that oneway functions exist ([ILL89], [H90]). (See also definition 1.4.9.)

Let (P, V) be any bounded NIZK proof system with polynomial time prover for the NP complete language L_R. We construct $(\bar{\mathrm{P}}\ , \bar{\mathrm{V}}\ )$, a general NIZK proof system for L_R, under the sole assumption that one-way functions exist.

Let $g : \{0,1\}^n \longrightarrow \{0,1\}^{2n}$ be a cryptographically secure pseudo-random bit generator. We divide the common random string σ into two segments: The first $2n$ bits which we call the *reference statement* y, and the rest of the common random string, which will be used as a reference string σ' for the bounded protocol (P, V). The reference statement is interpreted as "y is a pseudo-random string". This is an NP statement, whose witness is the n bit seed s which satisfies $g(s) = y$ (if such a seed exists). On input $(x, w) \in R$, prover $\bar{\text{P}}$ constructs a new NP statement $x\#y$. A witness to this statement is any string w' which either satisfies $(x, w') \in R$, or $g(w') = y$. $\bar{\text{P}}$ reduces $x\#y$ to an instance X of the NP complete language L_R, using a publicly known witness preserving reduction (known reductions have these properties). $\bar{\text{P}}$ reduces the witness w that he has for $x \in L_R$ to a witness for $X \in L_R$, and uses the system (P, V) and the reference string σ' to prove that $X \in L_R$.

Theorem 6.3.48 *Under the assumptions that (P, V) is a bounded NIZK proof system for L_R, and that oneway functions exist, the above transformed scheme is a general noninteractive zero knowledge proof system for L_R.*

Proof: We first give an intuitive introduction to the full proof. The completeness, soundness and zero knowledge properties of ($\bar{\text{P}}$, $\bar{\text{V}}$) are based on the corresponding properties of (P, V). The completeness property follows from the fact that from a witness to x, prover $\bar{\text{P}}$ can derive a witness to X, and thus execute the bounded NIZK proof system (P, V). The soundness property follows from the fact that y is chosen as a truly random (rather than pseudo-random) string. Thus, for almost all possible choices of y, $X \in L_R$ (allowing $\bar{\text{P}}$ to execute (P, V)) only if $x \in L_R$. The zero knowledge property requires more subtle analysis. The simulation of ($\bar{\text{P}}$, $\bar{\text{V}}$) is done by replacing the reference statement y by a pseudo-random string y'. This y' is generated by selecting at random an n bit seed s and computing $y' = g(s)$. Since g is a pseudorandom bit generator, this replacement is indistinguishable to polynomial time observers. Now any statement x with witness w is transformed into a statement X which also has s, the seed of y', as its witness. The simulator uses s instead of w in order to prove X. The concept of witness indistinguishability can now be used to show that this change of witnesses in the proof of X is indistinguishable to polynomial time observers, even if it is done polynomially many times. We now give a more detailed proof.

Completeness: The reduction to L_R preserves witnesses. Thus $\bar{\text{P}}$, who knows a witness for x, can also compute in polynomial time a witness for X, and use it in order to perform the protocol. The reduction is also publicly known, and so $\bar{\text{V}}$ can check that it was followed correctly. The completeness property then follows from the completeness property of (P, V).

Soundness: From the soundness property of (P, V) it follows that either $x \in L_R$, or y is pseudorandom (i.e., there is an s such that $y = g(s)$). But simple counting shows that the probability that the random string y of length $2n$ is pseudorandom (i.e., has a seed of length n) is at most 2^{-n}, and thus with overwhelming probability indeed $x \in L_R$.

The completeness and soundness property trivially continue to hold even if polynomially many statements are proved.

Zero knowledge: Assume $\bar{\mathrm{P}}$ proves membership in L_R of polynomially many inputs x_1, x_2, ... (using the associated witnesses w_1, w_2, ...). We construct a simulator M which creates an ensemble indistinguishable from the ensemble that $\bar{\mathrm{P}}$ produces.

M randomly selects an n bit seed s, and pseudo-randomly generates the $2n$-bits string $y' = g(s)$, to be used as the reference statement (instead of a random y used in reality). M generates a truly random reference string σ'. For each $j \geq 1$, M reduces $x_j \# y'$ to an instance X_j of L_R, and derives from s a witness w' for this instance. Then M uses the proof system (P, V) and the reference string σ' to simulate a proof that $X \in L_R$, by using his knowledge of the seed s rather than knowledge of the witnesses w_j to the statements x_j.

In order to prove that M's simulation is indistinguishable from $\bar{\mathrm{P}}$'s proofs, we construct a hybrid $\bar{\mathrm{M}}$, which constructs y' pseudo-randomly as M does, but gives the proofs using w_1, w_2, ... as $\bar{\mathrm{P}}$ does.

Lemma 6.3.49 *$\bar{M}$'s output is indistinguishable from $\bar{P}$'s output.*

Proof: Otherwise, $\bar{\mathrm{P}}((x_1, w_1), (x_2, w_2), ...)$ forms a nonuniform polynomial time statistical test for the pseudo-randomness of y, which is a contradiction. $\diamond$

We now prove that the outputs of M and $\bar{\mathrm{M}}$ are indistinguishable. Protocol (P, V) is zero knowledge. By Theorem 6.2.45 it is witness indistinguishable. By Theorem 6.2.47, even polynomially many executions of (P, V) on the same reference string are witness indistinguishable. $\bar{\mathrm{M}}$ and M differ only by the witnesses that they are using, and so by the witness indistinguishability property of the protocols, the ensembles that they create are indistinguishable. $\diamond$

Remark: In bounded NIZK proof systems, the length of the common random string bounds the size of the NP statement that can be proved. [BDMP89] show that with general NIZK proof systems statements of unbounded size can be proven. Their technique can be adapted to our case as well (details omitted).

6.4 Conclusions

The first step in proving that our general construction is zero knowledge was to replace the zero knowledge property of the bounded proof system by witness indistinguishability. Our task would have been easier had we started directly with a witness indistinguishable bounded noninteractive proof system rather than with a zero knowledge one. Consequently, in order to construct general NIZK proof systems, it is sufficient to construct a bounded

NIWI proof system. This may be an easier task than constructing a bounded NIZK proof system, because proving the WI property of a protocol does not involve the design of a simulator M.

Our construction also illustrates how a witness indistinguishable protocol can form the basis of a zero knowledge protocol. The idea is to add an auxiliary statement x' and have the prover prove a compound statement of the form "either $x \in L$, or x' is correct". Using the witness indistinguishability property of this proof, a simulator which knows a witness for x' (or for a statement which is indistinguishable from x') can simulate the compound proof, demonstrating that it is zero knowledge. This idea is used again in Chapter 8.

Chapter 7

Witness Hiding Protocols

7.1 Identification Schemes

An identification scheme is a protocol which enables party P to prove his identity polynomially many times to party V without enabling party V to later misrepresent himself as P to someone else. In [FFS87], zero knowledge proofs of knowledge were suggested as suitable candidates for identification protocols. P publishes a public key x and keeps secret a private key w satisfying $(x, w) \in R$, for some relation R testable in polynomial time. In order to prove his identity to V, party P gives a zero knowledge proof that he knows the private key associated with his public key. The zero knowledge identification protocol never discloses P's witness, and nobody can misrepresent himself as P unless he really knows P's witness (since the protocol is a proof of knowledge).

On taking a closer look at the security requirements of identification protocols, the authors observed that the use of zero knowledge protocols may be an overkill. Zero knowledge guarantees that no information whatsoever leaks during the execution of the protocol, but for identification protocols it is sufficient that V cannot extract P's private key, and we do not care whether V learns anything else (such as the size of P's shoes). This led [FFS87] to the formulation of the concept of *transferable information*. Using this relaxed, though still adequate, security condition, and special number theoretic properties, the authors construct simple constant round identification protocols, instead of the zero knowledge based protocols which require an unbounded number of rounds.

In this chapter we generalize and further develop the approach taken in [FFS87]. We introduce the concept of *witness hiding*, which informally means that the verifier cannot learn a witness to the common input from the proof of knowledge that the prover gives.

Using the concept of witness indistinguishability, we develop a methodology of constructing witness hiding protocols based on any NP language. Our protocols remain witness hiding even under parallel composition. These results are used in order to construct constant round identification protocols based on any oneway function. Additional applications of witness hiding protocols are presented in Chapter 8.

7.2 Witness Hiding

Informally, a protocol (P, V) is *witness hiding* (WH) if participating in the protocol does not help V to compute any new witnesses to the input which he did not know at the beginning of the protocol. In order to prove the WH property, one must show that if V' can compute a witness to the input after participating in the interactive proof, then he had this capability in him even before the protocol began. The definition of WH involves a probability distribution over the inputs. For this end, we borrow (and slightly modify) terminology from [AABFH].

Definition 7.2.50 *A random polynomial time algorithm G is a* generator *for relation R if on input 1^n it produces instances $(x, w) \in R$ of length n. G is an* invulnerable generator *if for any polynomial time nonuniform* cracking *algorithm C, Prob$((x, C(x)) \in R) < \nu(n)$, where $x = G(1^n)$. The probability is taken over the coin tosses of G and C.*

Definition 7.2.51 *Let (P, V) be a proof of knowledge system for relation R, and let G be a generator for this relation. (P, V) is* witness hiding *(WH) on (R, G) if there exists a witness extractor M which runs in expected polynomial time, such that for any nonuniform polynomial time V'*

$$\text{Prob}(V'_{P(x,w)}(x) \in w(x)) \quad < \quad \text{Prob}(M(x; V', G) \in w(x)) + \nu(n)$$

where $x = G(1^n)$. The probability is taken over the distribution of the inputs and witnesses, as well as the random tosses of P, V' and M. The witness extractor is allowed to use V' and G as blackboxes.

There are two main differences between WH and zero knowledge:

1. Witness hiding only guarantees that "whole" witnesses are not disclosed. Partial information may leak. In particular, the communication tape generated by $V'_{P(x,w)}(x)$ may not be simulatable in random polynomial time, and thus may serve as evidence that the protocol took place. In some cases this is an advantage. For example: Digital signatures cannot be zero knowledge (otherwise they are forgeable) and thus zero

knowledge is an inadequate framework for defining their security. On the other hand, digital signatures can be witness hiding, hiding the auxiliary information which allows the true signer to sign messages.

2. The distribution on the inputs enters the definition (through G). There might be infinitely many inputs on which P willingly discloses his witness, but the protocol may still be WH if the probability of G picking such an input is negligible. This distribution on the inputs also implies an important difference as to how V's nonuniformity manifests itself. In ZK protocols, V's auxiliary input may depend upon the actual common input x. In WH protocols, V's auxiliary input may depend only on n, the size of the common input, but the actual common input x is generated only after V's auxiliary input is fixed.

Remark: Goldreich [G89] presents a definition of zero knowledge which involves only inputs produced by probabilistic polynomial time generators. His goal is to treat known zero knowledge protocols in terms of uniform complexity measures. In contrast, our goal is to construct new protocols which are not known to be zero knowledge, and we treat them using nonuniform complexity assumptions. This leads to two differences in the role of the probabilistic polynomial time generator in the definitions: 1. We allow cheating verifiers to be nonuniform, whereas [G89] defines them as uniform algorithms whose auxiliary input is generated in probabilistic polynomial time. 2. Our definition allows a system to be WH with respect to some generators and not WH with respect to others, whereas in [G89] the system must be zero knowledge with respect to any probabilistic polynomial time generator.

Theorem 7.2.52 *Any zero knowledge proof system is also witness hiding with respect to any input distribution.*

Proof: Follows directly from the definitions. $\diamond$

In Chapter 4 we have seen that the parallel composition of zero knowledge (and thus of witness hiding) protocols may result in a protocol which is not witness hiding. Thus it is not true that in general WH is preserved under composition. If fact, the following simple example shows that WH is not preserved even under sequential composition.

Assume P, who knows a witness w for $x \in L$, picks at random an index i, where $1 \leq i \leq |w|$, and sends to V the index i together with the i^{th} bit of w. This procedure is witness hiding, for if V can compute w after this protocol, then he could have done so by himself by trying for each possible value of i the two possible values for the i^{th} bit of w (there are only polynomially many such cases). Now consider what happens if this protocol is repeated $O(|w|^2)$ times. Then almost surely each possible index i is sent to V at least once, and V

can completely reconstruct w. This sequential composition of witness hiding protocols is not witness hiding, unless finding a witness for x is in BPP.

7.3 Relation to Witness Indistinguishability

Our goal is to construct protocols for which the witness hiding property is preserved under arbitrary composition. We achieve this by relating the concept of witness hiding to that of witness indistinguishability. We cannot prove that any WI protocol is also WH, but we can specify simple conditions under which WI implies WH. These conditions involve the particular method by which input instances are generated. A protocol may be trivially WI if the relation R is such that every input has only one possible witness. In this case WI cannot imply anything. Furthermore, Theorem 4.2.32 demonstrates that in this case one should not trust even ZK protocols in nonsequential compositions. But if each input has at least two "computationally independent" witnesses, then the WI property is nontrivial, and it is possible to infer WH from WI.

One example of problems with computationally independent witnesses is that of families of "claw-free" functions [GoMiRi].

Definition 7.3.53 *Let $f^n : \{0,1\}^n \longrightarrow \{0,1\}^n$ be a function computable in time polynomial in n. Two arguments, x and x', satisfing and $f(x) = f(x')$ are called a* claw. *A family of functions $\{f_i^n\}$ is* claw-free, *if for any nonuniform polynomial time algorithm A, for sufficienly large n, given random input i, the probability that A outputs a claw for f_i^n is negligible. A family of claw-free functions is* proper, *if for any i, n, any image of f_i^n has at least two preimages.*

One example of a (conjectured) proper family of claw-free functions is squaring modulo a composite. Let i range over all composite integers of length n, and let $f_i^n(x) = x^2 \pmod{i}$. If the domain of f_i^n is restricted to integers smaller then $i/2$, then finding a claw (two independent square roots of the same argument) implies factorization of i, which is assumed to be intractable (for random i).

Theorem 7.3.54 *Let G be a generator for a proper family of claw-free functions: Given n, generator G selects i of size n at random, and generates pairs (x, w), where $x = f_i^n(w)$, with uniform distribution over the arguments w. Let (P, V) be a proof of knowledge system for proving knowledge of a pre-image of x with respect to f_i^n, where (x, i) is the common input to the proof system. Then if (P, V) is WI over $\{f_i^n\}$, then it is WH over $(\{f_i^n\}, G)$.*

Proof: Assume that $V'_{P((x,i),w)}((x,i),y)$ can output a preimage of x with respect to f_i^n

with nonnegligible probability n^{-k}. We show how a polynomial time algorithm M can find "claws" in random members of $\{f_i^n\}$ with nonnegligible probability. Given random i, M selects w' at random and computes $x' = f_i^n(w')$. Now M executes $V'_{P((x',i),w')}((x',i),y)$, performing P's publicly known polynomial time algorithm and using V' as a blackbox. With probability n^{-k}, V' outputs a preimage of x'. Since the protocol is WI, and x' has at least two preimages, the probability this preimage differs from w' is at least $1/2$. Thus M finds a claw in f_i^n with nonnegligible probability, contradicting the claw-freeness of $\{f_i^n\}$. $\diamond$

Claw free functions are rare. Many candidates for intractable functions do not even have two preimages (e.g. the discrete log). We show here a transformation which transforms any relation R to a new relation R^2 for which each argument has two independent witnesses.

Definition 7.3.55 *The* compound relation R^2 *associated with relation* $R = \{(x,w)\}$ *is defined by:*

$$((x_1,x_2),w) \in R^2 \text{ iff } (x_1,w) \in R \text{ or } (x_2,w) \in R$$

Given a generator G for R, obtain a generator G^2 for R^2 by applying G twice independently, and discarding at random one of the two witnesses.

Theorem 7.3.56 *Let G be a generator for relation R. Let (P,V) be a proof of knowledge system for R^2 (P proves knowledge of a witness of one of two instances in R). Then if (P,V) is WI over R^2, then it is WH over (R^2,G^2).*

The proof of this Theorem is quite complicated, and is given in Chapter 9. It is the only Theorem in this thesis whose proof uses the concept of *witness extractor*, introduced in Definition 7.2.51.

In typical cryptographic scenarios, it is assumed to be intractable to compute any witness from the common input alone. Under this assumption, the proof of Theorem 7.3.56 can be greatly simplified. Because of its cryptographic applications, we state this special case as a separate Theorem.

Theorem 7.3.57 *Let G be an invulnerable generator for relation R. Let (P,V) be a proof of knowledge system for R^2. Then if (P,V) is WI over R^2, then it is WH over (R^2,G^2).*

Proof: Assume the contrary. Then there exists V' whose probability of cracking an instance of R^2 after interacting with P is at least n^{-k}, for some integer k and infinitely many n. We construct M which has a non-negligible a-priori probability of cracking an instance of R (without interacting with P), thus contradicting G's invulnerability.

On input x, M uses G to generate an auxiliary solved instance (x_1, w_1). Now he uses the description of polynomial time P and his control over V' to run (P, V') on input (x, x_1), given in random order. M uses his knowledge of w_1 in order to perform P's part in the protocol. The probability that V' generates a witness to one of the two instances is n^{-k}. Because of the WI property, the witness V' produces is independent of the particular witness P is using, and so V' cracks x with probability $\frac{n^{-k}}{2} - \nu(n)$. Since all the algorithms involved (M, P and V') are polynomial time, this contradicts our assumption that G is an invulnerable generator. $\diamond$

In order to construct WI proofs of knowledge which are also WH, we composed two random instances of the NP language to derive a compound statement. This statement can be reduced to an instance of the NP-complete language of Hamiltonicity. P can use the zero knowledge Protocol 1.6.13 to prove knowledge of a witness to the compound statement. This protocol is also witness indistinguishable (Theorem 5.2.35). WI is preserved even if the basic steps are composed in parallel (Theorem 5.3.39). By Theorem 7.3.56, these parallel protocols are also witness hiding (on G^2).

Corollary 7.3.58 *Let G be a generator for relation R. Then under the assumption that one way functions exist, R^2 has a constant round proof of knowledge which is witness hiding over (R^2, G^2).*

Remark: Note a subtle point in the argument preceding the corollary. The witness hiding property is not preserved under parallel composition (see for example Theorem 4.2.32), but witness indistinguishability is preserved. Consequently, in proving that the parallel composition is witness hiding, we first prove that the composition is WI, and only then we deduce the WH property (with respect to generators of type G^2, which differ from the generators used in Theorem 4.2.32).

If an NP problem has random self reducibility properties [AL83], then its respective relation R has perfectly zero knowledge proofs of knowledge which do not depend upon unproven cryptographic assumptions [TW87]. In this case, it is undesirable to go through the general reduction to an NP complete problem in order to construct a protocol for R^2. Fortunately, this is not necessary, and one can construct constant round perfectly witness indistinguishable proofs of knowledge for R^2 relations based on random self reducible languages. We give an example based on the discrete logarithm problem. We modify this problem so that each instance of the new modified problem has two independent witnesses.

Protocol 7.3.59 *The common input is generated by a slight twist to the invulnerable generator of the discrete log assumption (see Definition 1.4.11), which forces the input to have* *two witnesses. The input is (p, g, c, x_1, x_2), where p, g, and c are as described in Definition 1.4.11, and x_1 and x_2 are integers in Z_p^* chosen randomly and independently. This compound instance is defined to have two possible witnesses: w_1 satisfying $g^{w_1} = x_1$ mod p, and w_2 satisfying $g^{w_2} = x_2$ mod p (w_i is the* discrete logarithm *of x_i). P receives as witness $w \in \{w_1, w_2\}$. P proves that he knows the discrete log of either x_1 or x_2. The basic step of the protocol is:*

1. *P chooses secretly, randomly and independently r_1, r_2, and computes $y_1 = x_1 g^{r_1}$ mod p, $y_2 = x_2 g^{r_2}$ mod p. P sends these two values in a random order to V.*

2. *V replies by a random challenge: 0 or 1.*

3. *If P receives 0, P reveals r_1 and r_2, and V checks that y_1 and y_2 were constructed correctly (satisfy $y = xg^r$ mod p). If P receives 1, P reveals the discrete log of* only one of *the y's (which equals $w + r$ (mod $p - 1$)).*

This basic step is executed $\log p$ *independent times* in parallel.

Theorem 7.3.60 *The above protocol is a complete and sound witness hiding proof that P knows the discrete log of one of two inputs, under the distribution of inputs specified above.*

Proof (sketch): The proof of the completeness and soundness properties of the protocol is standard (see Protocol 1.6.12 and Proposition 4.1.31), and is omitted. The proof of the witness hiding property demonstrates again the techniques of Theorem 7.3.57.

The basic step of the protocol is zero knowledge (and even perfect zero knowledge). The parallel composition of the basic step gives a protocol which is presumably not zero knowledge, but still (by Theorems 5.2.35 and 5.3.39) this protocol is WI. Assume that for some V', Protocol 7.3.59 is not WH. Then V' learns with nonnegligible probability one of the discrete logs (w.l.o.g., assume it is w_1). But because of WI, V' has the same probability of learning w_1 whether P is really using w_1 or the other witness w_2. This gives a random polynomial time algorithm for computing the discrete log. On input (p, g, c, x), M has to compute w satisfying $x = g^w$ mod p. M chooses randomly and uniformly w_2 and creates $x_2 = g^{w_2}$ mod p. M sets $x_1 = x$ and uses (p, g, c, x_1, x_2) as input to Protocol 7.3.59. M simulates the execution of this protocol by using w_2 to perform P's part and using his control over V' to obtain V's replies. By our assumption, V' has nonnegligible probability of extracting w_1 from this protocol. Thus M has nonnegligible probability of computing the discrete log of x, violating the discrete log assumption. $\diamond$

7.4 Conclusions

This chapter analyses how the prover's knowledge might leak in interactive proofs. We study intermediate cases between the two extreme categories, of zero knowledge proofs, and of proofs which can leak everything. We show that there is an interesting and useful intermediate category: Proofs which are not known to be zero knowledge, but which do not leak the prover's witness. We also initiate a case analysis of parallel composition of zero knowledge protocols: For statements which have only one witness, parallel composition may result in the disclosure of this witness, whereas for statements with two independent witnesses, this cannot happen.

Witness hiding proofs of knowledge are ideal candidates for identification schemes. By Corollary 7.3.58, we can construct parallel identification protocols based on any oneway function. A somewhat simpler construction can be based on any claw free function (by Theorem 7.3.54). Indeed, the efficient identification scheme described in [FFS87] is based on the number theoretic problem of squaring modulo a composite, which is assumed to be a claw free function.

Chapter 8

Zero Knowledge Arguments

8.1 Historical Overview

In zero knowledge interactive proof systems, the verifier is assumed to have "reasonable" computational resources (i.e., random polynomial time), whereas the prover may be "unreasonably" strong (e.g., exponential time). However, in cryptographic applications it is usually assumed that all participating parties have approximately the same computational resources and differ only in the knowledge that they have (e.g., knowledge of private keys). Thus, even though all P-space languages have interactive proof systems [S89] which can be performed in zero knowledge (under the assumption that oneway functions exist, see [BGGHKMR]), the more useful zero knowledge protocols are those for NP-statements (e.g., [GMW86]) in which the real prover need not be stronger than random polynomial time. Extending this observation, Brassard, Chaum and Crepeau [BCC88] suggested the notion of *zero knowledge arguments* in which even cheating provers are restricted to random polynomial time computation.

When considering proofs of knowledge, again it makes sense to restrict provers to polynomial time computations. Informally, we identify a machine's "knowledge" with whatever it can compute in polynomial time. If the prover (either truthful or cheating) is not restricted to polynomial time computations, this definition does not make much sense.

Having this restriction on the computational powers of cheating provers, Brassard, Chaum and Crepeau [BCC88] construct *perfect* zero knowledge arguments for any NP language, under the "intractability of the discrete log" assumption. This should be contrasted with Fortnow's proof that no NP complete language has a perfect zero knowledge interactive proof system unless the polynomial hierarchy collapses [F87]. In this Chapter we show another application of the extra flexibility offered by the restriction in power of cheating provers. We construct constant round zero knowledge arguments for any NP statement, based on the assumption that oneway functions exist. The best previously known such proof system relied on the stronger cryptographic assumption that clawfree permutations exist [GKa]. Before surveying the wealth of results connected with constant round proof systems, we clarify the terminology we use, since some authors use different terminology.

Definition 8.1.61 *A* move *of an interactive proof is a messages sent by one of the participants. Two moves (a message sent by V followed by a message sent by P) are called a* round.

The first attempted construction of constant round zero knowledge proofs for any NP language dates back to a remark in [GMW86] where the authors hint how to modify their unbounded rounds protocols and achieve a four move protocol. Alas, proving the correctness of the construction implied by this hint met unexpected technical difficulties, and was characterized by Goldreich (in a private communication) as an open problem. Subsequently, Goldreich and Kahan [GKa] slightly modified the construction to obtain a five move zero knowledge interactive proof for any NP statement, based on the assumption that claw free permutations exist.

Two constructions of constant round zero knowledge arguments for any NP language emerged independently. One is by Brassard, Crepeau and Yung [BCY89], based on the assumption that *oneway group homomorphisms* exist (the discrete log is the best known instance of an assumed oneway group homomorphism). The protocol takes six moves and achieves perfect zero knowledge. The other construction is the one which we present here (a preliminary version appeared in [FS89b]). Our protocol offers several levels of security, depending on the strength of the cryptographic assumption we make:

1. By relying only on the assumption that oneway functions exist, our protocol takes five moves and is computational zero knowledge.

2. By relying on the assumption that one-to-one oneway functions exist, our protocol takes only four moves, and is still computational zero knowledge.

3. By relying on the *discrete log assumption*, our protocol takes four moves, and is *perfect* zero knowledge.

The four moves protocols are optimal among all the zero knowledge protocols in which the zero knowledge property is proved by resettable blackbox simulation. This follows from

[GKr], where it is proved that only languages in BPP have 3 move interactive proofs which can be proven zero knowledge by such a simulation.

We now give an overview of our construction. Assume P wants to prove that he knows a witness w_p for input x_p. The protocol starts with V choosing a hard auxiliary statement x_v to which he knows a witness w_v. Now P, instead of giving a zero knowledge proof that he knows w_p, gives a witness indistinguishable proof that he knows some w' which is either a witness for x_p or a witness for x_v. This is just as good as a zero knowledge proof, because V can simulate P's proof in an indistinguishable way by using his knowledge of w_v instead of w_p which he does not know. To guard against cheating verifiers who might choose x_v without knowing a witness for it (e.g., choose $x_v = x_p$), V is requested to prove his knowledge of a witness for x_v before P gives his proof. To guard against cheating provers who do not actually know a witness for x_p, but instead try to extract w_v from V and then use it to give their proof, V's proof of knowledge must be witness hiding.

As we see, our zero knowledge protocol is composed of two subprotocols: A witness hiding subprotocol in which V proves knowledge of an auxiliary statement, and a witness indistinguishable subprotocol in which P proves knowledge of a witness to a compound NP statement composed of the common input and the auxiliary statement. Since we have seen (in Chapters 5 and 7) how to construct constant round witness hiding and witness indistinguishable protocols, it follows that the whole protocol can be carried out in a constant number of rounds.

Before presenting our protocol, we want to eliminate two possible sources of confusion:

1. Some languages (the random self reducible languages and their complements) have constant round zero knowledge protocols not relying on any cryptographic assumption ([GMW86],[BMO90]). Our protocols do rely on cryptographic assumptions, but they work for any NP language.

2. In Chapter 6 we discuss the concept of noninteractive (and thus one move) zero knowledge protocols. Noninteractive zero knowledge assumes the existence of a random string agreed upon by P and V, whereas our protocols start from scratch.

8.2 The Constant Round Protocol

We assume the existence of a one-way function f (see Definition 1.4.8). By [Naor89], this implies also the existence of bit commitment schemes (see Definition 1.4.10). Using subprotocols which are parallelizations of Protocol 1.6.13, we construct a constant round zero knowledge argument for the language of *directed Hamiltonian cycle* (DHC). Any other NP statement can be proved in zero knowledge by first reducing the statement to an instance of this NP complete language. This reduction must satisfy three properties:

1. It must be computable in polynomial time.
2. It must be *witness preserving.* This property is necessary for the completeness property of the protocol. It enables P, which has a witness to the original NP statement, to construct a witness to the generated instance of DHC.
3. Witnesses must be *efficiently invertible.* This property is necessary for the knowledge soundness property of the protocol. It allows V to conclude that P knows a witness to the original NP statement, even though P only demonstrates knowledge of a witness to the syntactically different NP complete statement.

We specify a polynomial reduction procedure from any NP language L to DHC, to be used whenever such a reduction is called for. This ensures that the reduction process itself conveys no information (and thus the zero knowledge property is preserved). We assume L is initially given as a nondeterministic Turing machine which accepts L. The reduction proceeds in two stages: Reducing the computation of this nondeterministic Turing machine to an instance of SAT (see [Cook71]), and reducing SAT to DHC (see [HS]). The reader may check for himself that this chain of reductions is witness preserving and efficiently invertible.

We now present the full protocol. An intuitive explanation of the protocol will follow, and the full proof of correctness will be given in the next section.

Protocol 8.2.62 *Let x denote the common input of size n and let w denote the NP witness that P has on its knowledge tape (i.e., $(x, w) \in R$ for some prespecified relation R). The protocol is composed of four moves, each one of which takes several steps.*

1. (a) V selects randomly and independently two values w_1 and w_2 in the domain of f, and computes $x_1 = f(w_1)$ and $x_2 = f(w_2)$. V reduces the NP statement "there exists w' such that either $f(w') = x_1$ or $f(w') = x_2$" to an instance G_V of the NP complete problem DHC. The witnesses w_1 and w_2 are transformed into two cycles in G_V, of which V randomly discards one and keeps the other (denoted as H_V). V sends (G_V, x_1, x_2).

(b) *Let n_v denote the number of nodes in G_V. V chooses randomly and independently n cycles, each on n_v nodes, and constructs the n respective adjacency matrices H_V^1, H_V^2, ... , H_V^n. V sends a secure commitment to each one of the entries of each one of the matrices.*

2. (a) *P checks that G_V is indeed obtained from (x_1, x_2) by the publicly known polynomial reduction. If this check fails, P stops.*

   (b) *P sends n random and independent bits b_p^1, b_p^2, ... b_p^n.*

   (c) *P reduces the NP statement "there exists w' such that either $(x, w') \in R$ or $f(w') = x_1$ or $f(w') = x_2$" to an instance G_P of the NP complete problem DHC. The witness w is transformed into a cycle H_P in G_P. P sends G_P.*

   (d) *Let n_p denote the number of nodes in G_P. P chooses randomly and independently n cycles, each on n_p nodes, and constructs the n respective adjacency matrices H_P^1, H_P^2, ... , H_P^n. P sends a secure commitment to each one of the entries of each one of the matrices.*

3. (a) *For each j, $1 \leq j \leq n$: If $b_p^j = 0$, V reveals all commitments to the entries of H_V^j. If $b_p^j = 1$, V sends P a permutation π^j that maps H_V^j onto the cycle H_V of G_V, and reveals the values of all entries in $\pi^j(H_V^j)$ which do not correspond to edges in G_V.*

   (b) *V checks that G_P is indeed obtained from (x, x_1, x_2) by the publicly known polynomial reduction. If this check fails, V stops and rejects.*

   (c) *V sends n random and independent bits b_v^1, b_v^2, ... b_v^n.*

4. (a) *For each j, $1 \leq j \leq n$: If $b_p^j = 0$, P checks that H_V^j indeed represents a Hamiltonian cycle. If $b_p^j = 1$, P checks that all the values of H_V^j that V revealed in step 3(a) are $'0'$ (nonedges), and that the locations that V did not reveal correspond to $\pi^j(G_V)$. If any of the checks fail, P stops.*

   (b) *For each j, $1 \leq j \leq n$: If $b_v^j = 0$, P reveals all commitments to the entries of H_P^j. If $b_v^j = 1$, P sends a permutation σ^j that maps H_P^j onto the cycle H_P of G_P, and reveals the values of all entries in $\sigma^j(H_P^j)$ which do not correspond to edges in G_P.*

5. *For each j, $1 \leq j \leq n$: If $b_v^j = 0$, V checks that H_P^j indeed represents a Hamiltonian cycle. If $b_v^j = 1$, V checks that all the values of H_P^j that P revealed in step 4(b) are $'0'$ (nonedges), and that the locations that P did not reveal correspond to $\sigma^j(G_P)$. If any of the checks fail, V rejects. Otherwise, V accepts.*

Protocol 8.2.62 is composed of two subprotocols whose moves are interleaved in such a way as to minimize the total number of moves. Each one of the subprotocols is the parallel version of Protocol 1.6.13, executed on a specially chosen input.

The first subprotocol is composed of steps 1(b), 2(b), 3(a) and 4(a). In this subprotocol the roles are reversed: V proves to P that he knows a cycle in a graph G_V. The graph G_V is constructed by V in step 1(a) by composing two random and independent instances of a difficult NP language based on the oneway function f. By Corollary 7.3.58, this subprotocol is witness hiding.

The second subprotocol is composed of steps 2(d), 3(c), 4(b) and 5. In this subprotocol P proves to V that he knows a cycle in a graph G_P, which is constructed in step 2(c). By Theorem 5.3.39, this subprotocol is witness indistinguishable.

The full protocol satisfies the three requirements of zero knowledge arguments of knowledge:

1. *Completeness:* Truthful V can execute the proof of knowledge for G_V since he can derive a cycle in G_V from his choices of preimages for f. Truthful P can execute the proof of knowledge for G_P since he can derive a cycle in G_P from his knowledge of w.

2. *Soundness:* P proves that he knows a cycle in G_P. This cycle must correspond either to a witness for the original NP statement, or to a preimage of one of the two instances of the oneway function that V picked. The probability of the second event is negligible, since P is polynomial time and cannot invert oneway functions. The fact that V proves knowledge of a preimage of one of the two instances does not help P to compute such a preimage because V's proof is witness hiding.

3. *Zero knowledge:* V's proof that he knows a cycle in G_V implies that he can also derive a cycle in G_P. The simulator M can use this fact to perform P's proof of knowledge of a cycle in G_P, by using the cycle that V derives instead of the cycle that P derives. M's execution would be indistinguishable from the real execution because P's proof is witness indistinguishable.

8.3 Proof of Correctness

Theorem 8.3.63 *Under the assumption that oneway functions exist, Protocol 8.2.62 is a zero knowledge argument of knowledge for any NP language.*

Proof: Protocol 8.2.62 is based on two executions of the parallel version of Protocol 1.6.13. The reader is referred to the discussion following the presentation of Protocol 1.6.13 and to Proposition 4.1.31 in order to complete details which are omitted from the proof given here.

Completeness: All reductions preserve witnesses. Thus truthful V who knows preimages both for x_1 and x_2 can derive from any one of them a cycle in G_V and use it to perform step 3(a). Likewise, truthful P can derive from w a cycle in G_P and use it in order to perform step 4(b).

Soundness: We exhibit an expected polynomial time knowledge extractor M which extracts from P a cycle in G_P. M follows moves 1 and 2 of the Protocol 8.2.62 exactly as V does. From move 3, M uses the knowledge extraction procedure described in the proof of Proposition 4.1.31. The analysis there proves that in expected polynomial time and with probability equal (up to negligible additive terms) to the probability that V accepts, M derives a cycle in G_P. (We note that the analysis is uneffected by the fact that in move 3(a) V reveals commitments from an earlier move of the protocol.) Using the fact that G_P was constructed by a reduction that is efficiently invertible, M transforms the cycle to a witness for the original composed problem (x, x_1, x_2). Now there are two possible cases:

1. There is negligible a-priori probability (before Protocol 8.2.62 begins) that the witness derived would be a pre-image under f of either x_1 or x_2.
2. This probability is non-negligible.

Assume that case 2 holds. With non-negligible a-priori probability, M extracts from P a pre-image of either x_1 or x_2. In order to derive a contradiction, we logically partition M into two algorithms: V which executes moves 1, 2 and 3(a), and M' which executes the rest of the knowledge extraction procedure. Now it follows that M' and P together can invert either x_1 or x_2 with non-negligible probability after interacting with V (but without ever resetting V). Even though M' is expected polynomial time, standard techniques (see Chapter 3) imply the existence of a nonuniform polynomial time algorithm which does the same. Since V's proof of G_V's Hamiltonicity is witness hiding (see Theorem 7.3.57 and Corollary 7.3.58), it follows that there exist a nonuniform polynomial time algorithm that can invert f with nonnegligible probability even without interacting with V, contradicting the assumption that f is oneway. This implies that case 1 holds, and consequently the witness that M extracts corresponds to the input problem, as desired.

Zero knowledge: We exhibit an expected polynomial time simulator M which simulates V's view of Protocol 8.2.62. M executes moves 1, 2, 3 and 4(a) of the protocol, performing

P's part of the protocol by himself (these steps do not require knowledge of a witness for the input statement), and letting V (who is treated as a blackbox) perform his part. If any of these steps is not completed successfully, M stops and the view created is perfectly indistinguishable from the real execution of the protocol. Otherwise, M is in the position that V completed a proof of knowledge of a cycle in G_V (steps 1(b), 2(b), 3(a) and 4(a)). Using the knowledge extractor of Proposition 4.1.31, M can now extract such a cycle from V. (We note that the execution of steps 2(d) and 3(c) which are irrelevant to V's proof of knowledge does not effect the proof of Proposition 4.1.31.) Since G_V was created by an efficiently invertible reduction, M can derive a preimage of either x_1 or x_2. Since the reduction to G_P preserves witnesses, M can derive a cycle in G_P and use it in order to execute step 4(b) and complete the protocol.

The view that M creates differs from the proof that the real prover gives only in the particular cycle in G_P that they use. But P's proof (steps 2(d), 3(c), 4(b) and 5) is known to be witness indistinguishable (see Theorems 5.2.35 and 5.3.39). This holds with respect to any common input and any auxiliary input to the verifier, and in particular with respect to the common input G_P and a verifier which has G_V and a cycle in it as auxiliary input. This verifier can perform by himself all steps of Protocol 8.2.62 which are not part of P's proof of knowledge of a cycle in G_P, and consequently also the full protocol is witness indistinguishable. It follows that M's simulation is indistinguishable from P's proof, as desired. $\diamond$

8.4 Discussion

Our use of WI and WH proof systems to construct zero knowledge protocols is a dual to Theorems 5.2.35 and 7.3.56 and to Corollary 7.3.58 which show that any zero knowledge proof system can be used to construct WI and WH proof systems.

In terms of the total number of bits transmitted, Protocol 8.2.62 is less efficient than Protocol 1.6.13 only by a constant factor. But in terms of the number of rounds, we have a dramatic improvement from n rounds in Protocol 1.6.13 to just a constant number in Protocol 8.2.62. This issue of the number of rounds deserves a closer look.

We note that step 5 of Protocol 8.2.62 is a check that V does on his own, and so it does not require the transmission of any message. Thus the total number of moves specified in the protocol is just four. But this count is a bit misleading, and actually depends upon the specific commitment schemes that V and P use in steps 1(b) and 2(d). The commitment schemes which can be used depend upon the strength of the cryptographic assumption that we make.

If we assume only the existence of oneway functions, then the only known construction of a commitment scheme that we can use is that of Naor [Naor89]. This commitment scheme requires an initialization message, which in our case would imply one more move.

Corollary 8.4.64 *Under the assumption that oneway functions exist, any NP based relation has a five move zero knowledge argument of knowledge scheme.*

If we assume the existence of one-to-one oneway functions, there are much simpler constructions of commitment schemes (e.g., see [GL89]). In this case the preliminary move is unnecessary.

Corollary 8.4.65 *Under the assumption that one-to-one oneway functions exist, any NP based relation has a four move zero knowledge argument of knowledge scheme.*

In both cases, the protocols are only computational zero knowledge. It is possible to further strengthen the cryptographic assumption and obtain a perfect zero knowledge protocol. Brassard, Chaum and Crepeau [BCC88] describe a commitment scheme based on the *discrete log* in which commitments to $'0'$ are perfectly indistinguishable from commitments to $'1'$. In an initial message V sends P an instance (p, g, c, x). P commits to a bit b (either $'0'$ or $'1'$) by choosing r at random and sending $g^r x^b \ (mod\ p)$. P reveals the value of the bit by sending r. The ability to open the same bit both as $'0'$ and as $'1'$ implies extracting the discrete log of x, which is assumed to be intractable. Commitments to $'0'$ are perfectly indistinguishable from commitments to $'1'$, since both are just random integers in the range $[1, p-1]$.

Corollary 8.4.66 *Under the* intractability of the discrete log *assumption, any NP based relation has a four move perfect zero knowledge argument of knowledge scheme.*

Proof: Add to Protocol 8.2.62 a step 1(c) in which V sends (p, g, c, x). P uses in step 2(d) the commitment scheme described above. There are two ways in which the proof of correctness of the protocol has to be modified.

Soundness: It is no longer true that each of P's commitments can be opened only in one possible way. Thus there is one more case to analyse in the knowledge extraction procedure: The case that M observes a violation of the soundness property of the commitment scheme. But from such a violation M can extract the discrete log of x. Since M, P and V are all polynomial time and do not know the discrete log of x to begin with, the probability of the latter event is negligible. Consequently, the soundness of Protocol 8.2.62 is not violated.

Zero knowledge: The simulation procedure for Protocol 8.2.62 starts with M trying to extract from V a cycle in G_V, and then M uses this cycle in order to complete the simulation. V's proof of knowledge satisfies the condition that if M obtains two successful executions which agree in step 1(b) (V's commitments) but differ in step 2(b) (M's requests) then M can extract a cycle in G_V. By the proof of Proposition 4.1.31 (which uses this condition), the probability that V convinces P that he knows a cycle in G_V and still M fails to extract a cycle from V is exactly 2^{-n}. Whenever this unlikely event happens, the expected polynomial time M should find a witness for x by exhaustive search, and complete the simulation. (We could ignore this event when our goal was just to achieve computational indistinguishability, but now our goal is to achieve perfect indistinguishability.) $\diamond$

We remark that this perfect zero knowledge protocol can be simplified in several ways.

1. Steps 1(b), 2(b), 3(a) and 4(a) are just a proof that V knows the discrete log of at least one of two instances. V's proof involves reducing this problem to an instance of DHC (step 1(a)), and using the parallel version of Protocol 1.6.13. This is unnecessary, as V can use Protocol 7.3.59 to prove his knowledge directly.

2. Step 1(c) in which V sends (p, g, c, x) as the basis of P's commitment scheme is redundant, since in step 1(a) V already sends (p, g, c, x_1, x_2). P can just use (p, g, c, x_2) as the basis of his commitment scheme.

3. In step 2(c), G_P is constructed by reducing the NP statement "there exists w' such that either $(x, w') \in R$ or $g^{w'} = x_1\ (mod\ p)$ or $g^{w'} = x_1\ (mod\ p)$" to an instance of DHC. It is sufficient to construct G_P from the simpler statement "there exists w' such that either $(x, w') \in R$ or $g^{w'} = x_1\ (mod\ p)$". This is justified because in proving the zero knowledge property, if M extracts the discrete log of x_1 from V he can complete the protocol by deriving a cycle in G_P, and if M extracts the discrete log of x_2 he can use it in order to simulate the protocol by violating the soundness property of P's commitment scheme.

A slightly modified version of this perfect zero knowledge protocol appears in [FS89b].

Chapter 9

Witness Hiding – Proof of the General Case

9.1 Introduction

In this chapter we prove Theorem 7.3.56 of Chapter 7. For the sake of completeness, we restate Definitions 7.2.50 and 7.3.55, and our main theorem:

Definition 9.1.67 *A random polynomial time algorithm G is a* generator *for relation R if on input 1^n it produces instances $(x, w) \in R$ of length n.*

Definition 9.1.68 *The* compound relation *R^2 associated with relation $R = \{(x, w)\}$ is defined by:*

$$((x_1, x_2), w) \in R^2 \text{ iff } (x_1, w) \in R \text{ or } (x_2, w) \in R$$

Given a generator G for R, obtain a generator G^2 for R^2 by applying G twice independently, and discarding at random one of the two witnesses.

Theorem 9.1.69 *Let G be a generator for relation R. Let (P, V) be a proof of knowledge system for R^2 (P proves knowledge of a witness of one of two instances in R). Then if (P, V) is witness indistinguishable over R^2, then it is witness hiding over (R^2, G^2).*

Before proving this theorem, we highlight the main ideas involved. Assume there is a possibly cheating verifier V which, after interacting with a truthful prover P that knows a witness to the input instance, has probability p' of cracking random instances of R^2. In order to prove that (P, V) is witness hiding over (R^2, G^2) we have to show how a witness extractor

M can use this V even without interacting with truthful P in order to crack random instances of R^2 with probability at least $p' - \nu(n)$. Note that we treat cracking probabilities as global properties over the input distribution created by G^2, and we do not insist that M succeeds on exactly the same input instances that V succeeds.

We first construct an algorithm C which cracks instances produced by G with probability $p = 1 - \sqrt{1-p'}$. Since G^2 produces instances by two independent applications of G, algorithm C can then be used in order to try to crack each of the two input instances independently, and the probability it cracks at least one of them is $2p - p^2 = p'$.

The construction of C is based on ideas already presented in the proofs of Theorems 7.3.54 and 7.3.57. In order to crack x, algorithm C uses G in order to generated an auxiliary solved instance $(x_1, w_1) \in R$. Now C simulates the execution of $V_{P((x,x_1),w_1)}(x, x_1)$ using V as a blackbox and running P's publicly available algorithm by himself. Since both x and x_1 were generated randomly and independently by G, then by our assumption V has probability p' of outputting a witness w' for (x, x_1). Since (P, V) is witness indistinguishable, w' is independent of w_1, and so $(x, w') \in R$ with probability $p'/2$.

In order to improve C's cracking probability up to $p = 1 - \sqrt{1-p'}$, the generator G is used to produce additional auxiliary instances, and the whole procedure is repeated. The main difficulty in the construction of C is in deciding exactly how many times C should invoke G before giving up: If C gives up too soon, he does not achieve cracking probability p, whereas if C continues for too long, his running time becomes superpolynomial. It turns out that the optimal number of times G has to be invoked is a function of p, but since p is unknown to C (which only receives V as a blackbox with no labels on it), C cannot compute this optimal number. This problem is solved by constructing a special procedure, called the "knockout procedure", which advises C online when to stop.

9.2 The Construction of the Cracking Algorithm

Let G be a generator for relation R, and let C denote algorithms which crack instances of R. Let C^2 denote algorithms which crack instances of R^2 generated by G^2.

Lemma 9.2.70 *There exists a uniform expected polynomial time algorithm C, such that for any C^2,*

$\text{Prob}((x, C(x; C^2, G)) \in R) > p - \nu(n)$

where $p = 1 - \sqrt{1-p'}$, and p' is the probability that C^2 cracks random instances of R^2. G and C^2 are given to C as blackboxes. In particular, C does not know which value of p it has to achieve.

Proof: We describe the behavior of C on input x of length n, generated by G:

Denote the following procedure by A: Apply $G(1^n)$ to obtain an auxiliary instance x_i. Apply the pair-cracking algorithm C^2 to the pair (x, x_i) (given in random order). A succeeds if $R(x, C^2(x, G(1^n)))$ holds. Denote by A_k k successive independent applications of A.

Let p_k denote the probability that A_k cracks x (for random x), where $p_0 = 0$. It is not trivial to compute this probability, since after A failed $k-1$ times, x is no longer from the original probability distribution G, but from G conditioned on $k-1$ previous failures.

Lemma 9.2.71

$$p - p_{(k+1)} \leq (p - p_k)\frac{p + p_k}{2} \tag{9.1}$$

Proof: By induction on k.

Base case: $k = 1$. A_1's cracking probability satisfies $p_1 \geq \frac{p'}{2} = \frac{2p-p^2}{2}$, which is also the probability obtained for p_1 by substituting $p_0 = 0$ in Equation 9.1.

Inductive step: For the sake of analysis we define a new algorithm $B_k(x, x_1)$, which tries to find witnesses to both of its inputs. $B_k(x, x_1)$ first invokes $A_k(x)$ in an attempt to find a witness for x, then invokes $A_k(x_1)$ in an attempt to find a witness for x_1, and finally invokes $C^2(x, x_1)$ as an attempt to crack either of the two inputs.

We identify three contributions to $E(B_k)$, the expected number of entries (from x and x_1) to which B_k produces a witness:

1. Both $A_k(x)$ and $A_k(x_1)$ succeed. By our induction hypothesis the probability of this event is p_k^2, and thus its contribution to $E(B_k)$ is $2p_k^2$.
2. $C^2(x, x_1)$ succeeds. Its contribution is $p' = 2p - p^2$.
3. A negative contribution due to the fact that the second contribution is redundant if the first contribution occurred. Since the probability of the first event is p_k^2, whereas the contribution of the second event is at most 1, this redundancy in the computation of $E(B_k)$ can be offset by a negative contribution of p_k^2.

It follows that $E(B_k) \geq 2p_k^2 + (2p - p^2) - p_k^2 = 2p - (p - p_k)(p + p_k)$. We are really interested in the event that B_k produces a witness for x. The probability of this event is at least $\frac{E(B_k)}{2} > p - (p - p_k)\frac{p+p_k}{2}$, which equals the desired value of $p_{(k+1)}$ in Equation 9.1. This probability does not change if we omit $B_k(x, x_1)$'s call to $A_k(x_1)$, since this last call is only concerned with deriving witnesses for x_1. But without this call, $B_k(x, x_1)$ is exactly $A_{(k+1)}$, since it makes just $k + 1$ calls to C^2. $\diamond$

If $p < 1$ is a constant independent of n, then A_n succeeds with probability $p - \nu(n)$, because each application cuts down the distance to p by a constant fraction (at least by p). But if the unknown p is $1 - o(1)$, we do not know how many times to repeat the procedure. To keep the *expected* running time polynomial, we want to repeat the procedure $\frac{n}{1-p}$ times. This timer is chosen so that we do not spend too much time on instances which are not solvable, and so that we allow sufficient time to solve instances which are solvable.

Lemma 9.2.72 $p_{\frac{n}{1-p}} \geq p - e^{-n}$

Proof: By Lemma 9.2.71:

$$p - p_{\frac{k+1}{1-p}} \leq (p - p_{\frac{k}{1-p}}) \prod_{i=\frac{k}{1-p}}^{\frac{k+1}{1-p}-1} (\frac{p + p_i}{2})$$

$$\leq (p - p_{\frac{k}{1-p}}) p^{\frac{1}{1-p}} \leq \frac{p - p_{\frac{k}{1-p}}}{e}$$

and the proof follows. $\diamond$

The expected running time of this procedure can be shown to be polynomial (the analysis is omitted). But how can C compute $\frac{n}{1-p}$? It may try to approximate $1 - p$ probabilisticly using $1 - p \simeq \sqrt{1 - p'}$, where p' is the observed success rate of C^2. But this method has a serious flaw: We cannot hope to get a meaningful estimate of p' before we observe at least one failure. The expected time until a failure is encountered is $\frac{1}{1-p'}$, and for very small $1 - p'$ this value can be much larger than $\frac{n}{1-p}$. Thus this method does not work in "real time".

In view of the above difficulty, we employ a different stopping procedure, which we call the "knockout" procedure. Let x_k denote the auxiliary instance generated by G on the k^{th} time it was invoked by C. C keeps a variable x' which is initialized as x_1. For any $k > 1$, if $C^2(x, x_k)$ fails to produce a witness for x, C also invokes $C^2(x', x_k)$. If $C^2(x', x_k)$ fails to produce any witness, C records this as an *interrupt* event, the current x' is discarded and x_{k+1} is later set as the new x'. If C^2 finds a witness for only one of the instances (x', x_k), the other instance is set as the new x'. If C^2 finds a witness to both instances, one of them randomly is set as the new x'.

Now we are ready to describe the complete expected polynomial time algorithm C. It involves three procedures, carried out in parallel, one step from each procedure at a time. The algorithm stops as soon as one of the procedures reaches its end.

Procedure 1: Use C^2 to crack x. Apply procedure A repeatedly, until $R(x, C^2(x, G(1^n)))$ holds (a witness for x is found).

Procedure 2: Exuastive search. Basic step - pick the next binary string y of length n and test $R(x, y)$. Procedure 2 ends when a witness for x is found. (At most 2^n steps).

Procedure 3: The knockout procedure. It ends when n interrupts are encountered.

Lemma 9.2.73 *C finds a witness for x with probability $p - \nu(n)$.*

Proof: It is sufficient to prove that with overwhelming probability, Procedure 3 does not stop in less than $\Omega(\frac{n}{1-p})$ steps. The result then follows from Lemma 9.2.72.

Procedure 3 counts n interrupt events. We show that with probability $> 3/4$, the number of calls to G between two successive interrupts is at least $\frac{1}{2(1-p)}$. Since after each interrupt all previous instances are discarded, the total number of steps is the sum of n independent events, and the proof of Lemma 9.2.73 follows from the Chernoff bound [Cher52].

Assume that $\frac{1}{2(1-p)}$ instances were generated since the last interrupt. They can be paired in $\binom{\frac{1}{2(1-p)}}{2} < \frac{1}{4(1-p)^2} = \frac{1}{4(1-p')}$ possible ways. By the linearity of the expectation, the expected number of times C^2 fails to crack a pair is $(1-p')\frac{1}{4(1-p')} = \frac{1}{4}$. Thus the probability that among $\frac{1}{2(1-p)}$ players there exists a pair that C^2 does not solve is smaller than $1/4$. $\diamond$

Lemma 9.2.74 *The expected running time of C is polynomial.*

Proof: Let $t(x)$ denote the random variable which counts the number of steps that C spends on a random input instance x. Procedure 2 guarantees that C runs for at most 2^n steps. Thus:

$$\sum_{i=1}^{2^n} Prob(t(x) = i) \ = \ 1$$

By definition, the expected running time of C is:

$$E(C) = \sum_{i=1}^{2^n} i Prob(t(x) = i)$$

We divide $E(C)$ into two contributions:

$$E(C) = E_0(C) + E_1(C)$$

where

$$E_0(C) = \sum_{i=1}^{n^2} i Prob(t(x) = i)$$

and

$$E_1(C) = \sum_{i=n^2+1}^{2^n} i Prob(t(x) = i)$$

Obviously, $E_0(C) \leq n^2$. Thus it is sufficient that we give a polynomial upper bound on $E_1(C)$. We first bound $E_1(C)$ from above by:

$$E_1(C) \leq \sum_{j=2\log n}^{n} 2^{j+1} Prob(t(x) > 2^j) \tag{9.2}$$

Consider any j, where $2\log n \leq j \leq n$, and assume $t(x) > 2^j$. Since, at most n interrupts of the knockout tournament occurred up to time 2^j, there must be some sequence of $\frac{2^j - n}{n}$ consecutive steps in which no interrupt occurred. Because of symmetry, in at least $1/n$ of the cases this sequence appears before the first interrupt occurs. Let $f(x)$ denote the first step in which an interrupt occurred. It follows that:

$$Prob(t(x) > 2^j) \leq n Prob(f(x) > \frac{2^j}{n} - 1) \tag{9.3}$$

Consider more closely the event $f(x) > k$. It implies that k auxiliary instances have been generated by Procedure 3, x has not been cracked, and no interrupt event occurred. This last fact implies that all (except possibly the current x') of the k auxiliary instances have been cracked. All instances (x, x_1, ..., x_k) where generated randomly and independently by G. By symmetry, the event that x is among the two elements which are not cracked has a-priori probability at most $\frac{2}{k+1}$. It follows that:

$$Prob(t(x) > 2^j) \leq n \frac{2n}{2^j}$$

and by Equation 9.2, $E_1(C) < 4n^3$. ◇

The proof of Lemma 9.2.70 follows from Lemmas 9.2.73 and 9.2.74. ◇

9.3 Proof of the Main Theorem

We first prove a simpler version of Theorem 9.1.69, in which (P, V) is assumed to be perfectly witness indistinguishable.

Theorem 9.3.75 *Let G be a generator for relation R. Let (P, V) be a proof of knowledge system for R^2. Then if (P, V) is perfectly witness indistinguishable over R^2, then it is witness hiding over (R^2, G^2).*

Proof: Let V be any possibly cheating verifier with auxiliary input y, and assume $V_{P((x_1,x_2),w)}((x_1, x_2), y)$ has cracking probability p' on random instances of R^2. (P, V) may be viewed as a cracking algorithm of type C^2. The construction of Lemma 9.2.70 gives a cracking algorithm C which has cracking probability $p - \nu(n)$ of cracking random instances x of R, where $p = 1 - \sqrt{1 - p'}$. This C uses G to generate auxiliary solved instances (x_k, w_k), and uses C^2 to crack the instances (x, x_k) of R^2. Based on this C, we construct a witness extractor M, as required by Definition 7.2.51 of witness hiding.

M has access to the blackboxes G^2, V and P. In order to imitate the action of C, the witness extractor must access G and C^2. M can simulate the action of G by applying G^2 and discarding the instance to which G^2 did not generate a witness. M can simulate $C^2(x, x_k)$ by executing $V_{P((x,x_k),w_k)}((x, x_k), y)$. Note that because of perfect witness indistinguishability, C^2's cracking probability is the same whether P is given a witness for x or a witness for x_k. Thus M can crack random instances of R with probability $p - \nu(n)$. In order to achieve cracking probability p' on inputs $(x_1, x_2) \in R^2$, machine M calls $C(x_1)$ and $C(x_2)$ independently, and the probability that one of the two inputs is cracked is at least $2p - p^2 - \nu(n) = p' - \nu(n)$.$\diamond$

The case of computational indistinguishability requires more careful analysis: The witness extractor M is expected polynomial time, and we have seen in Chapter 3 that the combination of expected polynomial time computations with computational indistinguishability can result in unexpected problems.

Proof of Theorem 9.1.69: Let V be any possibly cheating verifier with auxiliary input y, and assume $V_{P((x_1,x_2),w)}((x_1, x_2), y)$ has cracking probability p' on random instances of R^2. We show that minor modifications in the witness extractor M which is described in the proof of Theorem 9.3.75, result in a witness extractor which works also for the case of computational indistinguishability. We reprove two key Lemmas which are sufficient to prove our Theorem.

Lemma 9.3.76 *M's cracking probability is at least $p' - \nu(n)$ also in the computational indistinguishability case.*

Proof: In his attempt to extract a witness from V, the witness extractor M executes (P, V) on various pairs of instances. Since (P, V) is computational WI rather than perfectly WI, the cracking probability of $V_{P((x_i,x_j),w)}((x_i, x_j), y)$ may depend upon the actual witness that P is using (though in a very weak way). For the sake of analysis, we eliminate this dependency by constructing a new $\bar{\mathrm{M}}$, which always feeds P with a witness to the lexicographicly first of the two instances (x_i, x_j). Note that $\bar{\mathrm{M}}$'s executions of (P, V) differ from M's executions whenever the lexicographicly first instance happens to be one of the original input instances, since M does not know a witness to these instances. The proof of Theorem 9.3.75 works for this $\bar{\mathrm{M}}$, hence its cracking probability is $p' - \nu(n)$ and its expected running time is less than $4n^3 + n^2$.

Assume now that M's cracking probability is less than $p' - n^{-k}$, for some positive constant k. If we always stop M after at most n^{k+4} steps, his cracking probability only decreases. On the other hand, if we always stop $\bar{\mathrm{M}}$ after at most n^{k+4} steps, his cracking probability remains at least $p' - \frac{n^{-k}}{2}$, since the probability $\bar{\mathrm{M}}$ runs for more than n^{k+4} steps cannot be larger than $\frac{n^{-k}}{2}$ (for large enough n). Thus we have exhibited a polynomial difference of $\frac{n^{-k}}{2}$ between the probabilities of extracting witnesses by strictly polynomial time algorithms which differ only in the witnesses that the real prover is using, contradicting the witness indistinguishability property of (P, V). $\diamond$

Proving that M's expected running time is polynomial is inherently more difficult, as demonstrated by the following example. Assume that almost all the instances that G generates are easily solvable, but there is a negligible fraction $2^{\frac{-n}{2}}$ of instances which are inherently difficult. Assume furthermore, that whenever such an inherently difficult instance x_1 is paired with another instance x_2, $V_{P((x_1,x_2),w)}((x_1, x_2), y)$ cracks x_1 with probability $2^{\frac{-n}{8}}$ if $w \in w(x_1)$, and never cracks x_1 otherwise. Since $2^{\frac{-n}{8}}$ is negligible, this does not contradict the fact that (P, V) is computationally witness indistinguishable (though (P, V) cannot be perfectly witness indistinguishable). Now if the cracking algorithm C receives one of the difficult instances as its input instance, he would never crack it since C never applies (P, V) with a witness to the original input instance. Thus C would stop only when n interrupts of the knockout procedure occur.

Assume now that whenever the knockout procedure pairs two auxiliary instances to perform $V_{P((x_i,x_j),w)}((x_i, x_j), y)$, the witness it gives P is chosen at random from the two witnesses that M knows for x_i and x_j. Thus, whenever a difficult instance is generated as an auxiliary instance, it is expected to be knocked out of the procedure in $2^{\frac{n}{8}+1}$ steps. The probability that another difficult instance is generated during this time interval is $2^{\frac{-3n}{8}+1}$. An interrupt event is possible only when two difficult auxiliary instances are paired together. Consequently, the knockout procedure is expected to execute $\frac{n}{2}2^{\frac{7n}{8}}$ steps before n interrupt events occur. Thus the expected running time of C is at least $\frac{n}{2}2^{\frac{7n}{8}}2^{\frac{-n}{2}}$, which is exponential.

Lemma 9.3.77 *With slight modifications to M, its expected running time is polynomial.*

Proof: If M uses the cracking algorithm C carefully, then the fact that C is expected exponential time need not imply that M is expected exponential time. Rather than invoking $C(x_2)$ only after $C(x_1)$ ends, M can execute $C(x_1)$ and $C(x_2)$ in parallel, one step from each procedure at a time, and stop when the first of the two procedures stops.

Consider the effect of this parallel execution on the example above. In order for M to run for $\frac{n}{2}2^{\frac{7n}{8}}$ steps, both x_1 and x_2 must be difficult. The probability of this event is only 2^{-n}, making M's expected running time only one step.

In order to prove that parallel execution of $C(x_1)$ and $C(x_2)$ leads to expected polynomial time witness extraction in general (and not just in our specific example) we make another modification in M's use of C: Whenever M invokes $V_{P((x',x_k),w)}((x',x_k),y)$ on a pair of auxiliary instances, where x_k is the last auxiliary instance to be generated, the witness M gives P is the witness for x_k and not the witness for x'. Consequently, for each auxiliary instance x_i, $V_{P((x_i,x_j),w)}((x_i,x_j),y)$ is applied at most once with $w \in w(x_i)$.

As in the proof of Lemma 9.2.74, let $f(x)$ denote the first step in which an interrupt occurred. We want to bound the probability of the event $f(x) > k$.

Let K denote the set of the $k+1$ elements which were generated by G up to step k: The input element x and the auxiliary instances x_1 to x_k. Let S_i, for each i, denote the set of elements $x_j \in K$ for which the a-priori probability that $V_{P((x_i,x_j),w_j)}((x_i,x_j),y) \in w(x_i)$ is greater than $1/3$. We note that by computational witness indistinguishability, the a-priori probability that $V_{P((x_i,x_j),w_i)}((x_i,x_j),y) \in w(x_i)$ for $x_j \notin S_i$ is at most $1/3+\nu(n)$. Let $S \subset K$ be the set of all elements for which $|S_i| \leq 4n$.

For the sake of analysis, for each $x_i \in K$, consider a procedure that tries to crack it by executing $V_{P((x_i,x_j),w_j)}((x_i,x_j),y)$ once with each $x_j \in K$. Let B denote the set of x_i which were not cracked by this procedure. Consequently, each element $x_i \in B$ has only one opportunity of being cracked by the knockout procedure: In the first application of $V_{P((x',x_i),w_i)}((x',x_i),y)$. We distinguish between two cases:

1. $|B| \leq \sqrt{k}$: The input instance x is not cracked only if $x \in B$. The probability of this event is $\frac{|B|}{k+1} < \frac{1}{\sqrt{k}}$.

2. $|B| > \sqrt{k}$: For each $x_i \in K \setminus S$, the a-priori probability that it is not cracked by at least one of $V_{P((x_i,x_j),w_j)}((x_i,x_j),y) \in w(x)$, where $x_j \in S_i$, is at most $(\frac{2}{3})^{4n} < 2^{-2n}$. Since $k < 2^n$, the a-priori probability that there exists $x \in K \setminus S$ which is not cracked is at most 2^{-n}. Consequently, the a-priori probability that $B \subset S$ is $1 - 2^{-n}$. The contribution of the event $B \not\subset S$ is negligible and it is ignored.

   The probability that at least n elements of B are among the first $n^2\sqrt{k}$ instances generated by G is at least $1 - 2^{-n}$. For each of these elements x_i, the probability that at least one of $x_j \in S_i$ is generated among the first $n^2\sqrt{k}$ elements is $\frac{4n^3}{\sqrt{k}}$. Thus with probability $1 - \frac{4n^4}{\sqrt{k}}$, none of these x_i is paired with a member of the respective S_i in the first application $V_{P((x',x_i),w_i)}((x',x_i),y)$. Thus for each of them, the probability of being cracked at the first attempt is at most $1/3 + \nu(n)$. The probability that all but three of them are cracked in the first attempt is at most $(1/3 + \nu(n))^{-n+3}\binom{n}{3} < 2^{-n}$ for large enough n.

   We established that with probability essentially $1 - \frac{4n^4}{\sqrt{k}}$ at least three elements are not cracked during the first $n^2\sqrt{k}$ steps. Since only one of them is the input instance x itself, an interrupt event must occur.

From the above analysis it follows that the probability that $f(x) > k$ is at most $\frac{4n^4}{\sqrt{k}}$. Since M stops when the first of $C(x_1)$ and $C(x_2)$ stops, Equation 9.3 of Lemma 9.2.74 should be modified to:

$$Prob(t(x) > 2^j) \leq (nProb(f(x) > \frac{2^j}{n} - 1))^2$$

From Equation 9.2 in Lemma 9.2.74 it follows that $E_1(C) \leq 32n^{12}$, proving Lemma 9.3.77. ◇

The reader may check for himself (through the proofs of Lemmas 9.2.73 and 9.3.76) that also for M as described in the proof of Lemma 9.3.77 the cracking probability remains $p' - \nu(n)$. The proof of Theorem 9.1.69 now follows from Lemmas 9.3.76 and 9.3.77. ◇

9.4 Conclusions

This chapter establishes the relationship between the two concepts of witness indistinguishability and witness hiding. It shows that if the property of witness indistinguishability has nontrivial content, that is – input statements really have two independent witnesses, then any witness indistinguishable protocol is also witness hiding. Though this theorem has a fairly simple formulation, its proof turns out to be surprisingly complex. In particular, the value obtained as the knowledge extractor's expected running time is alarmingly high: $O(n^{12})$. However, the large expected running time is mostly an artifact of the analysis, which often sacrifices tighter bounds in favor of simplicity of proof.

Theorem 9.1.69 is included in this thesis mostly because of its aesthetic appeal, and because of its proof which examplifies the power and perils of expected polynomial time computation. For cryptographic applications, the much simpler proof of Theorem 7.3.57 usually holds, due to intractability assumptions made on the difficulty of cracking the input instances.

We conclude by listing a few topics for further research:

1. We described a particular methodology for constructing distributions of NP complete problems which have two independent witnesses (see Definition 9.1.68). Do these problems have independent witnesses also under simpler distributions? For example, consider the probability space $G_{n,p}$ of graphs with n nodes in which each possible edge exists with probability p independently of the other edges. Are witness indistinguishable proofs of Hamiltonicity witness hiding over $G_{n,p}$?

2. Witness indistinguishable protocols are also witness hiding, under suitable conditions. Do they also hide partial information about the witnesses, such as individual bits?

3. All our constructions of WI protocols are modifications (e.g., parallelizations) of known zero knowledge protocols. Construct a WI (or WH) protocol, where the construction is based on a different idea.

Bibliography

[AABFH] M. Abadi, E. Allender, A. Broder, J. Feigenbaum, L. Hemachandra, "On Generating Solved Instances of Computational Problems" *Advances in Cryptology – CRYPTO '88 Proceedings, Springer Verlag 403, pp. 297-310.*

[AL83] D. Angluin, D. Lichtenstein, "Provable Security of Cryptosystems: a Survey" *TR-288, Yale University, 1983.*

[B85] L. Babai, "Trading Group Theory for Randomness" *Proc. of 17th STOC, 1985, pp. 421-429.*

[BFL90] L. Babai, L. Fortnow, C. Lund, "Non-Deterministic Exponential Time has Two-Prover Interactive Protocols", *Comput Complexity 1 (1991), 3-40.*

[BG89] M. Bellare, S.Goldwasser, "New Paradigms for Digital Signatures and Message Authentication Based on Non-Interactive Zero Knowledge Proofs", *Advances in Cryptology – CRYPTO '89 Proceedings, Springer Verlag 435, pp. 194-211.*

[BMO90] M. Bellare, S. Micali, R. Ostrovsky, "Perfect Zero-Knowledge in Constant Rounds" *Proc. of* 22^{nd} *STOC 1990, pp. 482-493.*

[BGGHKMR] M. Ben-Or, O. Goldreich, S. Goldwasser, J. Hastad, J. Kilian, S. Micali, P. Rogaway, "Everything Provable is Provable in Zero Knowledge" *Advances in Cryptology – CRYPTO '88 Proceedings, Springer Verlag 403, pp. 37-56.*

[Blum86] M. Blum, "How to Prove a Theorem So No One Else Can Claim It" *Proc. of the International Congress of Mathematicians, Berkeley, California, USA, 1986, pp. 1444-1451.*

[BDMP89] M. Blum, A. De Santis, S. Micali, G. Persiano, "Non-Interactive Zero Knowledge" *MIT/LCS/TM-430.*

[BFM88] M. Blum, P. Feldman, S. Micali, "Noninteractive Zero Knowledge and its Applications" *Proc. of 20th STOC 1988, pp. 103-112.*

[BM84] M. Blum, S. Micali, "How to Generate cryptographically Strong Sequences of Pseudo-Random Bits" *SIAM Jour. on Computing, Vol. 13, 1984, pp. 850-864.*

[BHZ87] R. Bopanna, J. Hastad, S. Zachos, "Does co-NP have Short Interactive Proofs?" *IPL 25, May 1987, pp. 127-132.*

[BCC88] G. Brassard, D. Chaum, C. Crepeau, "Minimum Disclosure Proofs of Knowledge" *JCSS, Vol. 37, 1988, pp. 156-189.*

[BCY89] G. Brassard, C. Crepeau, M. Yung, "Everything in NP can be argued in perfect zero knowledge in a bounded number of rounds" *Proc. of 16th ICALP, Stressa, Italy, 1989.*

[BGKW88] M. Ben-or, S. Goldwasser, J. Kilian, A. Wigderson, "Multi Prover Interactive Proofs: How to Remove Intractability", *Proceedings, 20th Annual ACM Symp. on Theory of Computing, 1988, pp. 113-131.*

[Cher52] H. Chernoff, "A Measure of Asymptotic Efficiency for Tests of a Hypothesis Based on the Sum of Observations" *Annals of Math. Stat., 1952, Vol 23, pp. 493-509.*

[C87] A. Condon, "Computational Models of Games", *ACM/MIT press dissertation award series, 1989.*

[CL86] A. Condon, R. Ladner, "Probabilistic Game Automata" *JCSS Vol 36, 1988, pp. 452-489.*

[CL89] A. Condon, R. Lipton, "On the Complexity of Space Bounded Interactive Proofs", *Proc the 30th 29th Annual IEEE Symp. on Foundations of Computer Science, 1989, pp. 462-467.*

[Cook71] S. Cook, "The Complexity of Theorem Proving Procedures" *Proc. of 3rd STOC 1971, pp. 151-158.*

[DMP87] A. De Santis, S. Micali, G. Persiano, "Non-Interactive Zero-Knowledge Proof Systems" *Proc of CRYPTO-87, pp. 52-72.*

[DS88] C. Dwork, L. Stockmeyer, "Zero-Knowledge with Finite State Verifiers" *Advances in Cryptology – CRYPTO '88 Proceedings, Springer Verlag 403, pp. 71-76.*

[DY] A. De Santis, M. Yung, "Metaproofs (and their Cryptographic Applications)" *manuscript, 1990.*

[FFS87] U. Feige, A. Fiat, A. Shamir, "Zero Knowledge Proofs of Identity", *Journal of Cryptology, 1988, Vol. 1, pp. 77-94. Preliminary version appeared in Proc. of the 19th STOC, 1987, pp. 210-217.*

[FLS90] U. Feige, D. Lapidot, A. Shamir, "Multiple Non-Interactive Zero Knowledge Proofs Based on a Single Random String" *Proc. of* 31st *FOCS, 1990, pp. 308-317.*

[FS89a] U. Feige, A. Shamir, "Multi-Oracle Interactive Protocols with Space Bounded Verifiers", *Proc. of the 4th annual conference on Structures in Complexity Theory, June 1989, IEEE Computer Society Press, pp. 158-164.*

[FS89b] U. Feige, A. Shamir, "Zero Knowledge Proofs of Knowledge in Two Rounds", *Advances in Cryptology – CRYPTO '89 Proceedings, Springer Verlag 435, pp. 526-544.*

[FS89c] U. Feige, A. Shamir, "On Expected Polynomial Time Simulation of Zero Knowledge Protocols" *Proc. of DIMACS Workshop on Distributed Computing and Cryptography Princeton, NJ, October, 1989, J. Feigenbaum and M. Merritt (eds.) AMS and ACM (pubs.), pp. 155-160.*

[FS90a] U. Feige, A. Shamir, "Witness Indistinguishable and Witness Hiding Protocols", *Proc. of* 22nd *STOC, 1990, pp. 416-426.*

[FST88] U. Feige, A. Shamir, M. Tennenholtz, "The Noisy Oracle Problem", *Advances in Cryptology – CRYPTO '88 Proceedings, Springer Verlag 403, pp. 284-296.*

[Feld] P. Feldman, *private communication in [GS86].*

[FS86] A. Fiat, A. Shamir, "How To Prove Yourself: Practical Solutions to Identification and Signature Problems", *Proceedings of CRYPTO 1986, Lecture Notes in Computer Science no. 263, Springer Verlag 1987, pp. 186-194.*

[F87] L. Fortnow, "The Complexity of Perfect Zero-Knowledge" *Proc. of 19th STOC, 1987, pp. 204-209.*

[F89] L. Fortnow, "Complexity-Theoretic Aspects of Interactive Proof Systems"', *MIT/LCS/TR-447.*

[LFKN89] C. Lund, L. Fortnow, H. Karloff, N. Nisan, "The Polynomial Hierarchy has Interactive Proofs", *Proc. of FOCS 90, to appear.*

[GHY85] Z. Galil, S. Haber and M. Yung, "A Private Interactive Test of a Boolean Predicate and Minimum-Knowledge Public Key Cryptosystems", *Proceedings of FOCS 1985, pp. 360-371.*

[G88] O.Goldreich, "Towards a Theory of Average Case Complexity (a Survey)" *Technion, Computer Science Department, Technical Report 531, December 1988.*

[G89] O. Goldreich, "A Uniform-Complexity Treatment of encryption and Zero-Knowledge" *TR-568, Computer Science Dept., Technion, Haifa, Israel, 1989.*

[GGM86] O. Goldreich, S. Goldwasser, S. Micali, "How to Construct Random Functions" *Jour. of ACM, Vol. 33, No. 4, 1986, pp. 792-807.*

[GKa] O. Goldreich, A. Kahan, "Using Claw-free Permutations to Construct Constant-Round Zero Knowledge Proofs for NP" *in preparation.*

[GKr] O. Goldreich, H. Krawczyk, "On the Composition of Zero-Knowledge Proof Systems" *Proc. of* 17^{th} *ICALP, Springer Verlag, Vol. 443, 1990, pp. 268-282.*

[GL89] O. Goldreich, L. Levin, "A Hard-Core Predicate for all Oneway Functions" *Proc. 21st STOC 1989, pp. 25-32.*

[GMW86] O. Goldreich, S. Micali, A. Wigderson, "Proofs that Yield Nothing But Their Validity and a Methodology of Cryptographic Protocol Design", *Proc. 27th FOCS, 1986, pp. 174-187.*

[GMW87] O. Goldreich, S. Micali, A. Wigderson, "How to Play any Mental Game", 19^{th} *STOC, 1987, pp. 218-229.*

[GM84] S. Goldwasser, S. Micali, "Probabilistic Encryption" *JCSS, Vol. 28, No. 2, 1984, pp. 270-299.*

[GMR85] S. Goldwasser, S. Micali, C. Rackoff, "The Knowledge Complexity of Interactive Proof Systems", *SIAM J. Comput. Vol. 18, No. 1, pp. 186-208, February 1989. Preliminary version in Proc. of 17th STOC, 1985, pp. 291-304.*

[GoMiRi] S. Goldwasser, S. Micali, R. Rivest, "A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks" *SIAM Journal on Computing, vol. 17, No. 2, pp. 281-308.*

[GS86] S. Goldwasser, M. Sipser , "Arthur Merlin Games versus Interactive proof systems" *18th STOC, 1986, pp. 59-68.*

[H90] J. Hastad, "Pseudo-Random Generators under Uniform Assumptions" 22^{nd} *STOC, 1990, pp. 395-404.*

[HU79] J. Hopcroft, J. Ullman, "Introduction to Automata Theory, Languages, and Computation" *Addison-Wesley, 1979.*

[HS] E. Horowitz, S. Sahni, "Fundamentals of Computer Algorithms" *Computer Science Press, Computer Software Engineering Series, (pp. 526-529).*

[ILL89] R. Impagliazzo, L. Levin, M. Luby, "Pseudorandom Generation from Oneway Functions" 21^{st} *STOC, pp. 12-24, 1989.*

[K88] J. Kilian, "Zero-Knowledge with Log-Space Verifiers" *Proceedings, 29th Annual IEEE Symp. on Foundations of Computer Science, 1988, pp. 25-35.*

[LS90] D. Lapidot, A. Shamir, "Publicly Verifiable Non-Interactive Zero-Knowledge Proofs" *Proc. of Crypto 90.*

[L86] L. Levin, "Average Case Complete Problems" *SIAM Journal of Computing, 1986, Vol. 15, pp. 285-286.*

[L89] R. Lipton, "Recursively Enumerable Languages have Finite State Interactive Proofs", *Technical Report, Princeton University, 1989.*

[Naor89] M. Naor, "Bit Commitment Using Pseudorandomness" *Advances in Cryptology – CRYPTO '89 Proceedings, Springer Verlag 435, pp. 128-137.*

[NY90] M. Naor, M. Yung, "Public-key Cryptosystems Provably Secure Against Chosen Ciphertext Attacks" *Proc. of STOC90, pp. 427-437.*

[N89] N. Nisan, "Co-SAT has Multi-Prover Interactive Proofs", *e-mail announcement, November 1989.*

[Oren87] Y. Oren, "On the Cunning Power of Cheating Verifiers: Some Observations About Zero knowledge Proofs" *Proc. 28th FOCS, 1987, pp. 462-471.*

[P83] C. Papadimitriou, “Games Against Nature” *Proceedings, 24th Annual IEEE Symp. on Foundations of Computer Science, 1983, pp. 446-450.*

[PR79] G. L. Peterson and J. H. Reif, “Multiple-Person Alternation”, *Proceedings, 20th Annual IEEE Symp. on Foundations of Computer Science, 1979, pp. 348-363.*

[Pratt] V. Pratt, “Every Prime Has a Succinct Certificate” *SIAM J. Computing 4 (1975), pp. 214-220.*

[Reif79] J. Reif, “Universal Two Person Games of Incomplete Information” *Proceedings, 11th ACM Sypm. on Theory of Computing, 1979, pp. 288-308.*

[Reif84] J. Reif, “The Complexity of Two-Player Games of Incomplete Information” *JCSS Vol 29, 1984, pp. 274-301.*

[R88] J. Rompel, *personal communication, in [K88].*

[S89] A. Shamir, “IP=PSPACE”, *Proc. of* 31^{st} *FOCS, 1990, pp. 11-15.*

[TW87] M. Tompa, H. Woll, “Random Self-Reducibility and Zero Knowledge Interactive Proofs of Possession of Information” *Proc. 28th FOCS, 1987, pp. 472-482.*

[Yao82] A.C. Yao, “Theory and Applications of Trapdoor Functions” *Proc. of* 23^{rd} *FOCS, 1982, pp. 80-91.*