The (True) Complexity of Statistical Zero Knowledge

(Extended Abstract)

Mihir Bellare[*] Silvio Micali[†] Rafail Ostrovsky[‡]

MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139

Abstract

Statistical zero-knowledge is a very strong privacy constraint which is not dependent on computational limitations. In this paper we show that given a complexity assumption a much weaker condition suffices to attain statistical zero-knowledge. As a result we are able to simplify statistical zero-knowledge and to better characterize, on many counts, the class of languages that possess statistical zero-knowledge proofs.

1 Introduction

An interactive proof involves two parties, a prover and a verifier, who talk back and forth. The prover, who is computationally unbounded, tries to convince the probabilistic polynomial time verifier that a given theorem is true. A zero-knowledge proof is an interactive proof with an additional privacy constraint: the verifier does not learn why the theorem is true [11]. That is, whatever the polynomial-time verifier sees in a zero-knowledge proof with the unbounded prover of a true theorem x, can be approximated by a probabilistic, polynomial-time machine working solely on input x.

A statistical zero-knowledge proof (SZK proof) is one for which this approximate view cannot be distinguished from the true view even when given unbounded computational resources (This will be made more precise in §2).

Statistical zero-knowledge is indeed a strong privacy requirement and designing protocols to meet it is a formidable task. In fact, we do not have too many examples of languages possessing SZK proofs. Moreover, only few properties of this class of languages are known; most notably the ones proved in [8],[1].

In this paper, we, under a complexity assumption,

(1) Present a simpler condition on a language L which guarantees that L has a SZK proof, and thus reduce the complexity of designing SZK proofs, and
(2) Use this to prove various properties of the class of languages that possess SZK proofs.

The second result has the form that if some function is hard to compute, something about SZK proofs becomes easier. The argument thus has the flavor of a reduction "against-the-flow" (an argument of similar flavor is the one of Yao: if the discrete log problem is hard then RP is contained in sub-exponential time).

Given that statistical zero-knowledge is a computationally independent notion, it is somewhat strange that properties about it could be proved under a computational intractability assumption. We discuss this point in §3.3.

In proving the second result above, we actually exhibit a general paradigm for proving that the class of languages possessing SZK proofs has a given property. This paradigm is described in §4 and may be of independent interest.

2 Definitions

2.1 Probability Spaces and Algorithms

These notations and conventions for probabilistic algorithms are derived from [12] and further extended.

We emphasize the number of inputs received by an algorithm as follows. If algorithm A receives only one input we write "$A(\cdot)$"; if it receives two we write "$A(\cdot,\cdot)$",

[*] Supported in part by NSF grant CCR-87-19689.

[†] Supported in part by NSF grant DCR-84-13577 and ARO grant DAALO3-86-K-0171.

[‡] Part of this work was done at Boston University, Department of Computer Science, partially supported by NSF grant DCR-86-07492.

and so on.

If A is a probabilistic algorithm then, for any input i the notation $A(i)$ refers to the probability space which to the string σ assigns the probability that A, on input i, outputs σ.

If S is a probability space we denote by $\mathbf{P}_S(A)$ the probability that S associates to the set A. If A consists of the single element e we write $\mathbf{P}_S(e)$ rather than $\mathbf{P}_S(\{e\})$. We denote by $[S]$ the set of elements to which S assigns positive probability.

If $f(\cdot)$ and $g(\cdot, \cdots)$ are probabilistic algorithms then $f(g(\cdot, \cdots))$ is the probabilistic algorithm obtained by composing f and g (i.e. running f on g's output). For any inputs $x, y, \ldots$ the associated probability space is denoted $f(g(x, y, \cdots))$.

If S is a probability space then $x \leftarrow S$ denotes the algorithm which assigns to x an element randomly selected according to S (that is, x is assigned the value e with probability $\mathbf{P}_S(e)$) (in the case that $[S]$ consists of only one element e we write $x \leftarrow e$ rather than $x \leftarrow \{e\}$).

For probability spaces $S, T, \ldots$, the notation

$$\mathbf{P}(p(x, y, \cdots) : x \leftarrow S; y \leftarrow T; \cdots)$$

denotes the probability that the predicate $p(x, y, \cdots)$ is true after the (ordered) execution of the algorithms $x \leftarrow S$, $y \leftarrow T$, etc. The notation

$$\{ f(x, y, \cdots) : x \leftarrow S; y \leftarrow T; \cdots \}$$

denotes the probability space which to the string σ assigns the probability

$$\mathbf{P}(\sigma = f(x, y, \cdots) : x \leftarrow S; y \leftarrow T; \cdots) ,$$

f being some function.

If S is a finite set we will identify it with the probability space which assigns to each element of S the uniform probability $\frac{1}{|S|}$. (Then $x \leftarrow S$ denotes the operation of selecting an element of S uniformly at random).

We let PPT denote the set of probabilistic (expected) polynomial time algorithms.

2.2 Stastical Indistinguishability of Ensembles

Definition 2.1 The probability spaces E_1 and E_2 are *statistically indistinguishable within* ϵ if

$$|\mathbf{P}_{E_1}(T) - \mathbf{P}_{E_2}(T)| < \epsilon$$

for all $T \subseteq [E_1] \cup [E_2]$.

Definition 2.2 Let $L \subseteq \{0,1\}^*$. An ensemble with index set L is a collection $\{E(x)\}_{x \in L}$ of probability spaces, one for each $x \in L$.

Definition 2.3 The ensembles $\{E_1(x)\}_{x \in L}$ and $\{E_2(x)\}_{x \in L}$ are *statistically indistinguishable* (written $\{E_1(x)\}_{x \in L} \cong \{E_2(x)\}_{x \in L}$) if for any polynomial p there exists an n such that for all $x \in L$ of length at least n, the probability spaces $E_1(x)$ and $E_2(x)$ are statistically indistinguishable within $\frac{1}{p(|x|)}$.

2.3 Interactive Proof Systems and Zero Knowledge

Full definitions of interactive Turing Machines (ITMs) and protocols, interactive proof systems, and zero-knowledge appear in [11]. We only summarize the notation that we use.

The probability that (A, B) accepts the common input x is denoted

$$\mathbf{P}(\ (A, B) \text{ accepts } x\ ) ,$$

and the probability space of all conversations between A and B on input x is denoted $(A \leftrightarrow B)(x)$ (the probability in both cases is taken over the random tapes of both A and B).

Definition 2.4 Let (P, V) be an interactive protocol. If the function e satisfies

- For every $x \in L$,

$$\mathbf{P}(\ (P, V) \text{ accepts } x) \geq 1 - e(|x|) .$$

- For every ITM $\widehat{P}$ and every $x \notin L$,

$$\mathbf{P}(\ (\widehat{P}, V) \text{ accepts } x) \leq e(|x|) .$$

then we say that the protocol has error probability e with respect to the language L.

Definition 2.5 Suppose (P, V) has error probability e with respect to L. We call (P, V) an *atomic proof system* for L if $e(n) \leq \frac{1}{3}$. We call (P, V) a *proof system* for L if $e(n) \leq 2^{-n}$.

The view of the verifier during an interaction with the prover is everything he sees: that is, his own coin tosses and the conversation between himself and the prover. Accordingly we define

Definition 2.6 Let $(P, \widehat{V})$ be an interactive protocol and let $x \in \{0,1\}^*$. The *view* of $\widehat{V}$ on input x is the probability space

$$\mathit{View}_{(P,\widehat{V})}(x) = \{\ (R, C) : R \leftarrow \{0,1\}^{p(|x|)} ;$$

$$C \leftarrow (P \leftrightarrow \widehat{V}(R))(x)\ \} ,$$

where p is a polynomial bounding the running time of $\widehat{V}$.

Definition 2.7 An interactive protocol (P, V) is a *statistical zero knowledge* protocol (SZK protocol) for L if for every polynomial time ITM $\widehat{V}$ there exists a PPT algorithm $S_{\widehat{V}}(\cdot)$ such that $\{S_{\widehat{V}}(x)\}_{x \in L} \cong \{\mathit{View}_{(P,\widehat{V})}(x)\}_{x \in L}$ (this $S_{\widehat{V}}$ is called the *simulator*).

Other types of zero-knowledge are defined in [11]; namely computational and perfect. The weaker computational notion we will ignore in this paper. *Perfect* zero-knowledge means that the ensembles $\{S_{\widehat{V}}(x)\}_{x\in L}$ and $\{View_{(P,\widehat{V})}(x)\}_{x\in L}$ are more than statistically indistinguishable: they are actually equal. For our purposes, perfect zero knowledge is covered as a special case of statistical zero-knowledge.

When we say "zero-knowledge" in this paper we always mean statistical.

The machines P and V of the above definitions are referred to as the *prover* and the *verifier* respectively. We denote by IP the class of languages possessing interactive proofs and SZK the class of languages possessing statistical zero knowledge interactive proofs.

2.4 The Discrete Log Assumption

For p prime, $Z_p^* = \{1, 2, \ldots, p-1\}$ forms a cyclic group under multiplication mod p. For a generator g of this group the map $f_{p,g} : Z_p^* \to Z_p^*$ defined by

$$f_{p,g}(x) = g^x \bmod p\ .$$

is a permutation on Z_p^*.

We let $F(p)$ denote the prime factorization of $p-1$. We consider families of circuits $\{C_k\}$ which take as input a k bit prime p, a generator g of Z_p^*, $F(p)$, and a $y \in Z_p^*$, and output a $x \in Z_p^*$. The circuit family is of polynomial size if there exists a polynomial p such that the size of C_k is at most $p(k)$ for all k.

Discrete log Assumption: Let $\{C_k\}$ be a polynomial sized family of such circuits and p a polynomial. Then there is a n such that the for all $k \geq n$ the probability that $C_k(p, g, F(p), y) = f_{p,g}^{-1}(y)$ is $\leq \frac{1}{p(k)}$ when p is a randomly selected k bit prime, g is a generator of Z_p^*, and y is randomly selected from Z_p^*.

Note that the inverting circuit is given the prime factorization of $p-1$.

3 Main Theorem

Zero-knowledge protocols would be much easier to design under the restriction that the verifier did not deviate from his prescribed program. Our main theorem is that the simple condition that there exist a simulator for the honest verifier is enough, given a complexity assumption, to attain statistical zero-knowledge.

The idea of reducing the problem of security for arbitrary parties to the case of honest parties comes from the work in secure distributed computing [18],[15]. These protocols however had computationally bounded parties and only achieved computational zero-knowledge.

3.1 Honest Verifier Zero Knowledge

Honest verifier zero-knowledge is a much weaker notion than zero-knowledge. It only asks that there exist a simulator for the honest verifier V. That is,

Definition 3.1 An interactive protocol $(\overline{P}, \overline{V})$ for L is an *honest verifier statistical zero knowledge* protocol (honest verifier SZK protocol) for L if there exists a PPT algorithm $S(\cdot)$ such that $\{S(x)\}_{x\in L} \cong \{View_{(\overline{P},\overline{V})}(x)\}_{x\in L}$ (this S is called the *honest simulator*).

3.2 Main Theorem

Our main result is that given a suitable complexity assumption[†], honest verifier zero knowledge is in fact just as strong as zero-knowledge.

Theorem 3.2 Suppose L has an honest verifier statistical zero knowledge protocol. Then, under the discrete log assumption, L has a statistical zero knowledge protocol.

Our proof of this theorem actually establishes something much stronger. Below we list the properties we achieve.

Properties:

(1) Our proof is an effective transformation (what has been called in the literature a "compiler"): it takes an honest verifier statistical zero knowledge protocol $(\overline{P}, \overline{V})$ and returns a statistical zero knowledge protocol (P, V).

(2) Error probabilities are preserved: if $(\overline{P}, \overline{V})$ had error probability $\bar{e}$ with respect to L then so does (P, V). In particular, if $(\overline{P}, \overline{V})$ was an (atomic) proof system for L then so is (P, V).

(3) The cost in interaction is minimal: there is a constant c such that if $(\overline{P}, \overline{V})$ had $m(|x|)$ rounds on input x then (P, V) has $cm(|x|)$ rounds on the same input.

(4) The complexity of P is not much more than that of $\overline{P}$: P is a probabilistic polynomial time machine that uses $\overline{P}$ as an oracle.

(5) (P, V) is a black box simulation SZK protocol (see §4.2 for full definitions and discussion of this issue).

Note: We can show that our Theorem 3.2 also holds for zero-knowledge *arguments* (the model of [5],[6] in which the prover is restricted to polynomial time).

[†] In this paper we use the discrete log assumption. We have also, however, been able to prove the theorem with a factoring assumption.

3.3 Discussion and Comparison with Previous Work

The fact that we are achieving *statistical* zero knowledge although we have a complexity assumption is one of the novelties of this work that needs some elaboration.

We have defined zero-knowledge interactive proof systems via three Turing machines: prover, verifier, and simulator. In order to better understand the zero-knowledge notion, it is convenient to also imagine another Turing machine whom we will call the judge. The judge tries to tell apart the probability spaces $S_{\widehat{V}}(x)$ and $View_{(P,\widehat{V})}(x)$. More specifically, the judge sees a number of samples drawn at random from one of these probability spaces. His job is to determine which space they came from. Zero-knowledge means that the judge cannot tell the two probability spaces apart with any significant probability.

Intuitively, then, the zero-knowledge is stronger if it can tolerate more powerful judges.

In *computational* zero knowledge the judge is not allowed to be more powerful than PPT. Such zero-knowledge proof systems usually work by encrypting information securely based on a complexity assumption. The two probability spaces $S_{\widehat{V}}(x)$ and $View_{(P,\widehat{V})}(x)$ might actually consists of entirely different elements: for example the former might be encryptions of 1s while the latter is encryptions of 0s. No polynomially bounded judge can tell these apart. A computationally unbounded judge, however, certainly could.

Statistical zero-knowledge corresponds to the much stronger notion of allowing the judge any amount of computing power but restricting him to polynomial size samples to work on. Since the judge is not computationally restricted, it is not clear that complexity assumptions will help design such systems. We show, however, that they do. Our simulator works by generating exactly the correct conversation except on a small set of bad instances on which it fails, as opposed to computational zero knowledge proofs, where *every* simulated instance may be different from any real conversation.

Statistical zero-knowledge *arguments* have been designed before using complexity assumptions [5],[6]. In this model, however, the prover is restricted to be PPT and the assumption is used against the prover. That is, if the prover can invert a one-way function, he can convince the verifier that some $x \in \overline{L}$ belongs to L. We stress that in our case the prover need not be computationally bounded.

4 Applications

Theorem 3.2 yields a simple and effective two step paradigm for proving properties of the class of statistical zero-knowledge languages:

- Step 1: Show that a SZK protocol has the property with respect to the honest verifier
- Step 2: Transform this protocol via Theorem 3.2, arguing that the transformation preserves the property. The result is a SZK protocol with the property, under the discrete log assumption.

We believe that this paradigm has wide applicability. Below we present four examples which will clarify the methodology.

4.1 Bounding the Complexity of the Prover

The prover is a function which given a partial conversation computes his next message. As a function, he has a certain complexity.

Paul Feldman observed that every language in IP possesses an interactive proof system whose prover runs in PSPACE. As SZK $\subseteq$ IP it might then seem that a PSPACE prover is enough for a SZK proof. But in fact the ZK constraint may require a prover to be more powerful.

So how powerful should a prover be for giving a ZK proof? This question was raised by Joe Kilian.

Feldman actually showed that a *deterministic* PSPACE prover suffices for IP. However Oren [17] showed that coin tosses are necessary: only for languages in BPP do there exist SZK proof systems with deterministic provers. A natural question then is whether probabilistic PSPACE is enough. We show (under a complexity assumption) that this is indeed the case.

The prover's complexity should not be confused with the complexity of the underlying language. Thus the fact that a language possessing a statistical zero-knowledge proof system lies in AM $\cap$ co-AM [8],[1] does not say anything about the complexity of a SZK prover for it.

The first step in our solution is

Lemma 4.1 Suppose $(P', \overline{V})$ is an honest verifier SZK protocol for L. Then there exists a probabilistic PSPACE prover $\overline{P}$ such that $(\overline{P}, \overline{V})$ is also an honest verifier SZK protocol for L.

Proof: Let S' be the honest simulator for $(P', \overline{V})$. In order to compute his response to a message from $\overline{V}$ we have $\overline{P}$ use S' to sample the space of all responses consistent with the current partial conversation. Details will be given in the final paper. □

Remark: Both the error probability with respect to L and the number of rounds remain the same for $(\overline{P}, \overline{V})$ as they were for $(P', \overline{V})$.

Theorem 4.2 Suppose L has an (honest verifier) SZK protocol. Then, under the discrete log assumption, L has a SZK protocol (P, V) with P of complexity probabilistic PSPACE.

Proof: By Lemma 4.1, L has an honest verifier SZK protocol $(\overline{P}, \overline{V})$ with $\overline{P}$ being of complexity probabilistic PSPACE. By Theorem 3.2 L then has a SZK protocol (P, V) with P (by Proposition 4; see §3.2) being probabilistic PSPACE. □

Remark: By properties 2 and 3 of Theorem 3.2 and the above remark, the error probability with respect to L is the same for (P, V) as it was for the original protocol while the number of rounds increases by only a constant factor.

4.2 The Power of Black Box Simulation

The definition of zero-knowledge stipulates that for each (cheating) verifier there exist a simulator: the simulator $S_{\widehat{V}}$ for $\widehat{V}$ is allowed to depend arbitrarily on $\widehat{V}$.

A simpler but more stringent condition would be to require a single simulator for all verifiers. To simulate the view of a specific verifier $\widehat{V}$ this simulator would use $\widehat{V}$ as a black box, setting $\widehat{V}$'s random tape, giving it inputs, and receiving outputs. This is known as black box simulation; we will define it more precisely below.

In principle it may be necessary to have a different simulator for each verifier. A surprising fact, however, is that black box simulation suffices for all known zero-knowledge proofs. This leads us to ask whether any SZK language has a SZK proof of this form. That is the question we address in this section.

Oren [17] formalized the black box notion by saying that the simulator was a PPT oracle machine M who when asked to simulate a particular verifier $\widehat{V}$ was given that verifier as an oracle:

Definition 4.3 An interactive protocol (P, V) for L is a *black box simulation statistical zero knowledge* (black box simulation SZK) protocol for L if there exists a PPT oracle TM $M(\cdot)$ such that for every polynomial time ITM $\widehat{V}$ it is the case that $\{M^{\widehat{V}}(x)\}_{x \in L} \cong \{View_{(P,\widehat{V})}(x)\}_{x \in L}$ (this M is called the *black box simulator*).

One has to be a little careful, however, about how one defines the running time of the oracle machine M. M's running time is a function only of his input x and is supposed to be a fixed polynomial p in the length of this input. Suppose $\widehat{V}$ used $q(|x|)$ coins on input x and $q > p$. How can M even output a string of $q(|x|)$ bits as part of his simulation? In order to overcome this technical difficulty we will let M have two random tapes. The first is used by M in the usual fashion and the second by $\widehat{V}$. M can reset $\widehat{V}$ on its random tape, and can also, in one step, output the entire string to the left of $\widehat{V}$'s position on this tape.

Under the discrete log assumption we show that black box simulation is not a restriction on zero-knowledge:

Theorem 4.4 Suppose L has an (honest verifier) SZK protocol. Then, under the discrete log assumption, L has a black box simulation SZK protocol.

Proof: By assumption L has an honest verifier SZK protocol. By Theorem 3.2 (Proposition 5; see §3.2) L then has a black box simulation SZK protocol. □

The remark made following Theorem 4.2 again applies.

4.3 One-Sided Zero Knowledge

A question that has attracted much research in complexity theory is whether *one-sidedness* is a restriction for probabilistic computation. For polynomial time computation the question is whether RP = BPP, and it remains open.

In the interactive scenario, on the other hand, Goldreich, Mansour and Sipser [13] show that one-sidedness is not a restriction. They define

Definition 4.5 An atomic proof system (P, V) for L is *one-sided* if $\mathbf{P}(\ (P, V) \text{ accepts } x) = 1$ for all $x \in L$.

and prove that any language in IP has a one-sided atomic proof system.

But is one-sidedness a restriction when zero-knowledge is involved? Goldreich, Mansour and Sipser prove their theorem by showing how to transform a given proof system for L into another which is one-sided. The transformation does *not* preserve zero knowledge: even if the original protocol had been zero knowledge, the transformed one may not be. The question that remained (and was raised in [13]) was: suppose L has a SZK proof system. Then does L have a SZK proof system which is one-sided?

As our next application we answer this question in the affirmative, under a complexity assumption. We use the fact that the transformation of [13] *does* preserve statistical zero-knowledge with respect to the honest verifier:

Lemma 4.6 [13] Suppose L has an honest verifier SZK atomic proof system. Then L has an honest verifier SZK one-sided atomic proof system.

and our general transformation then enables us to derive

Theorem 4.7 Suppose L has an (honest verifier) SZK (atomic) proof system. Then, under the discrete log assumption, L has a SZK one-sided atomic proof system.

Proof: By Lemma 4.6 L has an honest verifier SZK atomic proof system $(\overline{P}, \overline{V})$. By Theorem 3.2 L then has a SZK atomic proof system with (by Proposition 2; see §3.2)

$$\mathbf{P}(\,(P,V) \text{ accepts } x) = \mathbf{P}(\,(\overline{P},\overline{V}) \text{ accepts } x) = 1$$

for all $x \in L$. □

4.4 Parallelization

The core of almost all known zero-knowledge proof systems consists of an atomic zero-knowledge proof system which is then repeated serially, many times, in order to reduce the error probability to $2^{-|x|}$. This serial repetition maintains zero knowledge. The question of whether such a price in rounds must be paid in order to reduce the error probability while maintaining zero-knowledge has attracted a great deal of research effort.

For computational zero-knowledge, it has been shown that the zero-knowledge does not require a great price in rounds: one has constant round proofs for NP and a constant plus $m(n)$ round proofs for languages with $m(n)$ round proof systems [9], [16].

For statistical and perfect zero-knowledge however, the research has concentrated on designing round efficient zero-knowledge protocols for particular problems. For example [3] provides constant round perfect zero-knowledge proofs for any random self-reducible language such as graph isomorphism or quadratic residuosity. In the theorem that follows we, on the other hand, provide a general mechanism for parallelizing statistical zero-knowledge proof systems.

Theorem 4.8 Suppose L has an (honest verifier) SZK atomic proof system of $m(n)$ rounds. Then, under the discrete log assumption, L has a SZK proof system of $O(m(n))$ rounds.

Proof: Let $(\overline{P}, \overline{V})$ be the protocol which on input x consists of $6|x|$ versions of the original SZK atomic proof system run in parallel. Now observe that $(\overline{P}, \overline{V})$ is an honest verifier SZK proof system for L and is of $m(n)$ rounds. By Theorem 3.2 L then has a SZK protocol which (by Proposition 2) is a proof system for L and (by Proposition 3) has $O(m(n))$ rounds. □

We actually prove something much stronger. Namely, we provide a mechanism for running several (potentially different) zero-knowledge protocols in parallel, under the assumption that the discrete log problem is hard. That is, given a collection of zero-knowledge protocols we show how to compile the entire collection into a single protocol which essentially executes all the original protocols side by side. Thus, whenever many ZK proofs are used as primitives in some other protocol, our technique allows us to significantly reduce the number of rounds of interaction.

We stress that the above results also apply to the model of zero-knowledge *arguments* of [5],[6].

5 Proof of Main Theorem

5.1 An Overview

Our solution essentially *forces* the verifier to be honest. The very basic idea is for the *verifier* to prove to the *prover*, in "zero knowledge," that he is doing at every stage exactly what the honest verifier would do. Simulating this transformed protocol involves a combination of using the simulator for the honest verifier and simulating the overhead.

What gives a cheating verifier more power? It is not just the fact that he can deviate from the program of the honest verifier and execute his own protocol. A more subtle point is that instead of using unbiased coin flips from his random tape, he may use differently distributed coin flips. To overcome these difficulties, we begin by modifying the prover:

(1) the prover will force the verifier to use truly random bits. An immediate difficulty is that the prover must not know the contents of the random tape of the verifier. How can he force him to use truly random bits, without the knowledge as to what they are? The idea is to use something analogous to "coins flips into the well" protocol [4]. However the original "coin flips into the well" protocol does not work for our purposes and we introduce a modification of it. The extra complication which we encounter here, is the fact that the conversation must be simulatable in statistical Zero-Knowledge.

(2) After the prover has "blindly" provided the verifier a truly random tape, the verifier must essentially "prove" to the prover that he is "behaving honestly". That is, he must convince the prover not only that he is running the program of the honest verifier, but also that he is using the random tape provided to him by the prover. Needless to say, the verifier must nonetheless be assured that the prover cannot cheat. Moreover, the "proving to the prover" stage must also be simulatable in statistical zero-knowledge.

At first glance, it seems that we are making our life even more difficult – we are making the prover more complicated, thus also making the job of the simulator potentially harder. This is actually not the case. The extra complexity of the protocol will be carefully exploited by the simulator.

Clearly, our simulator would have to use the simulator for the honest verifier as a subroutine. The simulator for the honest verifier requires a random tape. Giving the random tape which the cheating verifier is using will not work. The extra complication introduced in the previous step now comes to our rescue: our simulator will behave like the prover and "blindly" provide a random tape to the (cheating) verifier. Then, "rewinding" the cheating verifier, the simulator will be able to *extract* the random tape provided "blindly" in the previous step. Now, our simulator will be able to call the simulator for the honest verifier as a subroutine, with the knowledge of the coin tosses the cheating verifier must use. The intuitive idea is that the cheating verifier will now be forced to behave like an honest verifier on the random tape known to our simulator. Thus, our simulator will perform careful interaction between the cheating verifier and an honest simulator and construct an almost perfect (i.e. statistical) simulation of the desired conversation.

5.2 Witness Independent Proofs for NP

Our protocol requires the ability of the PPT verifier to give "proofs of knowledge of a witness" to the infinitely powerful prover.

Why are the verifier's proofs to the prover useful? If the prover is infinitely-powerful and the verifier is poly-bounded, clearly the verifier cannot help the prover to compute anything that the prover could not compute by himself: after all, the infinitely powerful prover can compute anything he wants. However, what the prover may not know is the state of knowledge of the verifier. In essence, using witness-independent proofs, the verifier can convince the prover that he knows a witness to some NP problem without revealing which witness it is that he knows, out of all the possible ones.

For $x \in L$ let W_x denote the set of witnesses for x with some *fixed* representation. We then define

Definition 5.1 Let (A, B) be a bounded prover A, infinitely powerful verifier B interactive proof system for L which also demonstrates possession of information. We say that (A, B) is *witness independent* if for all (*computationally unbounded*) $\hat{B}$, for all $x \in L$, and for all $w_1, w_2 \in W_x$ we have

$$View_{(A(w_1),\hat{B})}(x) = View_{(A(w_2),\hat{B})}(x) \ .$$

Here $A(w)$ denotes the interactive Turing machine A which begins with w on its private tape.

For our purposes we require that the protocol be of a constant number of rounds and simulatable from A's point of view. Another requirement is that the simulator should be capable of extracting the exact witness which A is trying to hide.

We note that [7],[5] and [6] present essentially the same notion. Although they state it with respect to a PPT verifier, it can be extended to work with unbounded verifiers. Note that in our case, the verifier plays the role of A and the prover plays the role of B!

To implement a witness independent protocol, we note that it is enough to execute in parallel zero-knowledge proofs for NP statements in *fixed* number of envelopes. We stress that this protocol is *not* necessarily zero knowledge in the usual sense. This, however will not harm our simulation, since the directionality is reversed. We will elaborate more on this in the final paper.

5.3 Our Protocol

We give just the basic intuition of the proof of Theorem 3.2. Given $(\overline{P}, \overline{V})$, we have to construct another prover/verifier pair (P, V) such that

- (P, V) is also an interactive proof system for L
- For *any* (cheating) verifier $\widehat{V}$ there exists a simulator $S_{\widehat{V}}$.

Our protocol begins with P dealing V a secret random string. That is, P does not know the string but can ensure that it is random. However, we do this in such a way that the simulator can later control this random string. We proceed as follows, where x of length n is the common input, and $t(n)$ is a polynomial which bounds the running time of $\overline{V}$:

Step 1:

P generates a random prime number p together with the prime factorization of $p-1$. He does this via Bach's algorithm [2]: he keeps picking random numbers in factored form until he gets one whose successor is a prime. He then picks a generator g of Z_p^* and a random element $s \in Z_p^*$. he sends p, g, s and the prime factorization of $p-1$ to V.

Step 2:

V checks that p is prime, and using the prime factorization of $p-1$ he checks that g is a generator. He now picks a sequence $a_1, \ldots, a_t$ of random bits, and commits to them. He does this by picking $w_i \in \{0, \ldots, p-2\}$ at random, and for each i sending the prover

$$commit(a_i) = z_i = \begin{cases} g^{w_i} \bmod p & \text{if } a_i = 0 \\ sg^{w_i} \bmod p & \text{if } a_i = 1 \end{cases}$$

This committal gives P no information since z_i is random regardless of a_i. On the other hand, V cannot later decommitt to a value other than the one he committed without finding the discrete log of s.

Step 3:

At this point, we add a "dummy" step. We have V give P a *witness independent* proof that he "knows" how to decommit each of the z_i, without revealing any information about the committed bits. This basic idea appeared first in [11] in the context of quadratic residuosity. The proof, however, must be in constant rounds, and must be from the polynomially bounded verifier to the infinitely powerful prover. The crucial fact about *witness independent* proof, is that the simulator will be able to extract a witness (i.e. decommital of the committed bits) with overwhelming probability, while it reveals absolutely nothing to the prover.

Step 4:

P now picks t bits $b_1, \ldots, b_t$ at random and sends them to V. V lets $c_i = b_i \oplus a_i$ for each $i = 1, \ldots, t$. His secret random string is then

$$C = \boxed{c_1 c_2 c_3 \ldots c_t}\ .$$

The string C is unknown to P because, as pointed out before, he learns nothing from the committals. Moreover, if either P or V is honest, the string is random.

The next stage of the protocol consists of running the old $(\overline{P}, \overline{V})$ protocol (with V playing $\overline{V}$ and P playing $\overline{P}$) for L on input x, with two modifications:

- V, in running $\overline{V}$, uses as coins the secret random string C that he was dealt above
- Every message sent from V to P is accompanied by a *witness independent* proof that $\overline{V}$ would really have sent this message if his coins were C.

That is, V begins by sending the message α_1 that would have been the first message $\overline{V}$ sent on coins C, and proves that he did indeed do this. The prover checks this proof, and if it is incorrect he aborts. Otherwise he sends whatever response β_1 the old prover $\overline{P}$ would have sent. And so on.

The witness independent proof that $\widehat{V}$ sends needs some elaboration. For example, consider the first message α_1 that he sends P. The statement he will prove, written out in all its gory detail, is:

$$\begin{aligned} &\exists w_1 \ldots \exists w_t \;\; \exists a_1 \ldots \exists a_t \;\; \exists c_1 \ldots \exists c_t \\ &\Big[\textstyle\bigwedge_{i=1}^{t} ((z_i = g^{w_i}) \wedge (a_i = 0)) \\ &\qquad\qquad \vee ((z_i = sg^{w_i}) \wedge (a_i = 1)) \\ &\quad \wedge \textstyle\bigwedge_{i=1}^{t} (c_i = b_i \oplus a_i) \;\wedge \overline{V}(c_1 c_2 \ldots c_t, \lambda) = \alpha_1 \;\Big] \end{aligned}$$

5.4 The Simulator

We now describe the simulator. Recall that by the hypothesis of Theorem 3.2 we are given a simulator $S_{\overline{V}}$ for the interaction between $\overline{P}$ and $\overline{V}$. Let $\widehat{V}$ be any cheating verifier for the above protocol. We must now construct a simulator $S_{\widehat{V}}$.

The first step of this simulator is to run $S_{\overline{V}}$ to obtain a pair $(\overline{C}, \alpha_1\beta_1 \ldots \alpha_m\beta_m)$ consisting of $\overline{V}$'s coin tosses $\overline{C} = \bar{c}_1\bar{c}_2 \ldots \bar{c}_t$ and the transcript $\alpha_1\beta_1 \ldots \alpha_m\beta_m$ of the conversation between him and $\overline{P}$ in which that latter had $\overline{C}$ as coin tosses.

$S_{\widehat{V}}$ now fixes $\widehat{V}$'s random tape; that is, he fills it with coin tosses from his own random tape. He will now proceed to generate a simulation of the first stage of the protocol in such a way that the secret random string dealt to $\widehat{V}$ will end up being the string $\overline{C}$ he obtained above. He does this as follows:

(1) He acts as P would for step 1

(2) He runs $\widehat{V}$ for step 2 to get his committals $z_1, \ldots, z_t$.

(3) At this point, the simulator uses the "dummy" step to *learn* the values $w_1, \ldots, w_t$. Informally, he must first run the entire proof once, and then back up and run it again. It is not a trivial task to show that this leads to an expected polynomial time algorithm which extracts the the witnesses with overwhelming probability, but it can be done with arguments similar to those used for [14]'s zero knowledge proof of graph non-isomorphism.

(4) Having the w_i, the simulator can compute the a_i. He now picks $b_i = a_i \oplus \bar{c}_i$ for all $i = 1, \ldots, t$ as being the prover's response of step 4, and has thus succeeded in his goal of having $\overline{C}$ be the secret random string.

We now come to the stage of the simulation in which we really capitalize on the honesty we have enforced on the verifier. Recall that the simulator has in his possession the conversation $\alpha_1\beta_1 \ldots \alpha_m\beta_m$ of $\overline{P}$ and $\overline{V}$ when the latter's coins are $\overline{C}$. He runs $\widehat{V}$ and gets what is supposed to be $\overline{V}$'s first message if he had $\overline{C}$, together with a proof that this is indeed the case. He examines the proof and if it is incorrect he aborts as the prover would have. But if not, then with very high probability, the message $\widehat{V}$ sent is *really* the message α_1 that the simulator expected at this stage. And this message he can respond to: he just has to send β_1.

Continuing in this way the simulator soon has a transcript of the entire conversation.

Why is this statistical zero knowledge? The reason is that the simulator generates exactly the correct conversation except on a certain set of bad instances on which he fails. The latter could happen either if $\widehat{V}$ managed to find the discrete log of s or if he was able to cheat the prover in a witness independent proof. The set of bad instances thus has a total probability that is smaller than the reciprocal of any polynomial in $|x|$. Notice how the way in which the hardness of discrete log is used differs from the way it is used in [14] where *every* simulated instance differs from the real conversation.

Full proofs will appear in the final paper.

Acknowledgements

We thank Oded Goldreich for helpful comments.

References

[1] Aiello W., and J. Hastad "Perfect Zero-Knowledge can be Recognized in Two Rounds" FOCS 87.

[2] Bach, E., "How to Generate Factored Random Numbers," *SIAM Journal on Computing* **17**(2), 179-193 (April 1988).

[3] Bellare, M., S. Micali and R. Ostrovsky, "Perfect Zero-Knowledge in Constant Rounds," STOC 90.

[4] Blum, M., "Coin Flipping over the Telephone," IEEE COMPCON 1982, 133-137.

[5] Brassard, G. and C. Crépeau, "Non-transitive Transfer of Confidence: A perfect Zero-knowledge Interactive protocol for SAT and Beyond," FOCS 86.

[6] Brassard, G., C. Crépeau, and M. Yung, "Everything in NP can be Argued in Perfect Zero Knowledge in a Bounded Number of Rounds," ICALP 89.

[7] Feige, U. and A. Shamir, "Witness Indistinguishability and Witness Hiding," STOC 90.

[8] Fortnow, L., "The Complexity of Perfect Zero-Knowledge" STOC 87.

[9] Goldreich, O. and A. Kahn, personal communication.

[10] Goldreich, O., and H. Krawczyk, "On the Composition of Zero Knowledge Proof Systems," ICALP 90.

[11] Goldwasser, S., S. Micali, and C. Rackoff, "The Knowledge Complexity of Interactive Proofs," *SIAM J. Comput.,* **18**(1), 186-208 (February 1989).

[12] Goldwasser, S., S. Micali, and R. Rivest, "A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks," *SIAM J. Comput.,* **17**(2), 281-308 (April 1988).

[13] Goldreich, O., Y. Mansour, and M. Sipser, "Interactive Proof Systems: Provers that never Fail and Random Selection," FOCS 87.

[14] Goldreich, O., S. Micali, and A. Wigderson, "Proofs that Yield Nothing but their Validity", FOCS 86.

[15] Goldreich, O., S. Micali and A. Wigderson, "A Completeness Theorem for Protocols with Honest Majority," STOC 87.

[16] Naor, M. and M. Yung, "Universal One-Way Hash Functions and their Cryptographic Applications," STOC 89.

[17] Oren Y., "On The Cunning Power of Cheating Verifiers: Some Observations About Zero Knowledge Proofs", FOCS 87.

[18] Yao, A.C., "How to Generate and Exchange Secrets," FOCS 86.