J. Cryptology (1998) 11: 1–27

Journal of
CRYPTOLOGY

An Efficient Noninteractive Zero-Knowledge Proof System for NP with General Assumptions*

Joe Kilian
NEC Research Institute, 4 Independence Way,
Princeton, NJ 08540, U.S.A.
joe@research.nj.nec.com

Erez Petrank
Department of Computer Science, University of Toronto,
Toronto, Ontario, Canada M5B 1A4
erez@cs.toronto.edu

Communicated by Oded Goldreich

Received 20 November 1995 and revised 7 October 1996

Abstract. We consider noninteractive zero-knowledge proofs in the shared random string model proposed by Blum et al. [5]. Until recently there was a sizable polynomial gap between the most efficient noninteractive proofs for NP based on general complexity assumptions [11] versus those based on specific algebraic assumptions [7]. Recently, this gap was reduced to a polylogarithmic factor [17]; we further reduce the gap to a constant factor. Our proof system relies on the existence of one-way permutations (or trapdoor permutations for bounded provers).

Our protocol is stated in the *hidden bit model* introduced by Feige et al. [11]. We show how to prove that an n-gate circuit is satisfiable, with error probability $1/n^{O(1)}$, using only $O(n \lg n)$ random committed bits. For this error probability, this result matches to within a constant factor the number of committed bits required by the most efficient known *interactive* proof systems.

Key words. One-way permutations, Zero knowledge, Efficient proofs, Noninteractive zero knowledge, Circuit satisfiability.

1. Introduction

A basic issue in cryptography is the tradeoff between resources and security properties. Ordinary zero-knowledge proofs obtain greater security at the price of requiring greater interaction between the prover and the verifier. Blum et al. [5] propose a way to eliminate interaction while preserving the zero-knowledge property. Instead of requiring that the prover and verifier interact with each other, they only require that there be some common

* Most of this work was done while Erez Petrank was visiting the NEC Research Institute.

public random string which the prover and verifier have access to. The properties of the proof (completeness, soundness, and zero knowledge) depend on this shared string indeed being uniformly selected at random. Their model is known as the *shared random string model* for noninteractive zero-knowledge proofs.

In this model we enjoy the full security of zero knowledge and still the prover can send a zero-knowledge proof of a theorem as a single message to the verifier who can then check the proof without further interaction.

Such noninteractive zero-knowledge proofs turned out to be an important cryptographic primitive. For example, it is used in the signature scheme framework of [2] and in obtaining secure public-key encryption schemes that are robust against chosen message attacks [20]. This motivates the questions of what assumptions are necessary for noninteractive zero-knowledge proofs and how efficient can these proofs be made?

We also mention that another avenue of eliminating interaction was suggested by Fiat and Shamir [12]. They suggest a heuristic means for eliminating the need for interaction, at a cost of slightly weaker security properties. Namely, instead of showing a "zero knowledge simulation" of their protocol (which would mean that no knowledge leaks in the execution of the protocol) they show that an efficient algorithm for breaking their scheme implies an efficient algorithm to factor numbers. Although the security property is somewhat weaker, the security analysis is rigorous.

1.1. *Previous Results*

Blum et al. [5] showed how to construct a noninteractive zero-knowledge proof system for any language in NP given a specific number-theoretic assumption. This assumption was relaxed to quadratic residuosity by De Santis et al. [9]. (In their joint journal version [4], they also suggest how to prove "many theorems" while saving in the length of the random string.) The original solution required n^4 shared random bits to prove that an n-node graph is four-colorable, but the efficiency of protocols based on quadratic residuosity has been recently greatly improved [7], [6]. Given a cryptographic security parameter k, and an allowed error probability ε, the most efficient of these families of solutions requires $O(nk \log(n/\varepsilon))$ shared random bits.

Feige et al. [11] gave the first noninteractive zero-knowledge for NP under a general complexity assumption. Their construction requires one-way permutations or, for polynomially bounded provers, certified trapdoor permutations (the technical "certification" requirement was eliminated in [3]). They also obtained much greater generality in how their proofs could be used (e.g., many provers could prove many theorems using the same random string). To achieve these results they introduced the *hidden bits* model; all subsequent progress using general complexity assumptions has used this model. In the hidden bits model (sometimes called the *random committed bit model*), the prover and verifier are dealt a sequence of random bits, but only the prover is allowed to see the bits. The prover may reveal a subset of these shared bits to the verifier, and in addition he sends a message to the verifier. The verifier receives the message, the indices of the bits he may look at, and the actual bits that appear in these indices. Based on this information, the verifier chooses whether to accept or reject.

This model may seem strange at first sight, but seems more amenable for designing protocols. Assuming the existance of one-way permutations we can translate a protocol

designed for the hidden bits model into a protocol for the shared random string model [11]. Roughly speaking, the compilation procedure goes as follows: Each consecutive k bits in the shared random string can be considered by the prover and verifier as a commitment to a bit in the following way. The k bit-string s is interpreted as a commitment to the bit $B(P^{-1}(s))$ where P is the one-way permutation and B is a hard-core bit of the one-way permutation P. The prover may see what is behind the committed bits since he is all powerful, but the polynomial time verifier can only see bits that the prover "lets" him see. Namely, bits for which the prover sends the verifier the inverse of the corresponding string s. See Section 3 for more details.

The proof system proposed by Feige et al. [11] was much less efficient than those based on quadratic residuosity: it required $O(k \cdot n^{11/2} \cdot \log n \cdot \log(1/\varepsilon))$ shared random bits to prove that an n-node graph contains a Hamiltonian cycle with error probability at most ε. Subsequently, Feige [10] has shown how to reduce the required number of bits by a factor of n^2, thus using only $O(k \cdot n^{7/2} \cdot \log n \cdot \log(1/\varepsilon))$ shared random bits. We stress that (unless otherwise mentioned) the complexity is stated in the standard model, and the multiplicative term k originates from compiling a proof in the hidden bits model to the standard (shared random string) model.

Recently, a more efficient proof system for NP was given by Kilian [17] who presented a proof system for circuit satisfiability using $O(k \cdot n \cdot \log^c n \cdot \log(n/\varepsilon))$ shared random bits for some constant c. In addition to the efficiency of this proof, circuit satisfiability seems to be the preferred NP-problem for reducing a general predicate. Kilian's construction is based on the hidden bits model, and hence is based on the same complexity assumptions as those of [11].

1.2. *Our Results*

We improve the protocol of [17] and get a noninteractive zero-knowledge proof system for circuit satisfiability which uses only $O(kn \log(n/\varepsilon))$ shared random bits. Our proof system is significantly simpler than the one in [17]. As in [11] and [17] we also rely on the general cryptographic assumption that there exists a one-way permutation, or trapdoor permutations (though at a loss of efficiency; see the remark) if the prover only runs in polynomial time.

In the hidden bits model, we show how to prove circuit satisfiability using $O(n \lg(n/\varepsilon))$ random committed bits. For the case of error probability $\varepsilon = 1/n^{O(1)}$, this matches the most efficient *interactive* protocol known: we do not know how to achieve this error probability in the interactive model without using $\Omega(n \lg n)$ bit commitments.

In the above we measure efficiency in terms of the number of shared hidden bits used, l. We remark that the time efficiency is closely related but is determined by the complexity of the one-way permutation used. More precisely, the verifier has to perform computation which involves at most l invocations of the one-way permutations plus a computation which is "almost" linear in the l.[1]

[1] Almost may mean at most a logarithmic multiplicative factor. This extra factor appears when the model of computation is a Random Access Machine which performs basic operations in time which is linear in the *length* of the operands, rather than in constant time.

A remark about efficient provers. We also make a remark about the case of efficient provers since we consider this an important (practical) aspect of the protocol. In this case the compilation procedure of [11] requires a "cost" in the soundness error probability. Specifically, this error probability increases by a factor of 2^k. (For the details see Section 3.) Thus, in this case protocols (in the hidden bit model) have to be design with error probability ε substantially lower than 2^{-k}. Hence, for some combinations of the soundness error probability and the security parameter, noninteractive zero-knowledge proofs remain asymptotically less efficient than their interactive counterparts. In the interactive case no connection has to be made between the soundness error probability and the security parameter. (We refer the reader to [18], [1], and [16] for the state of the art in efficient zero-knowledge *interactive* proofs.)

1.3. *Outline of the Paper*

In Section 2 we introduce some definitions and technical details that we use in the paper. In Section 3 we recall the *hidden bits model* of [11] and the compilation from this model to the standard *shared random string* model. In Section 4 we give a short introduction to our approach and a top-down preview of the proof system. In Section 5 we give a bottom-up full description of the proof system. In Section 6 we prove that the proof system is valid, and in Section 7 we explain what modifications are needed in the proof system so that the prover can be implemented efficiently.

2. Preliminaries

We set the notations and definitions we use. A parametrized ensemble of distributions over a language L is a set $E = \{E(x,k) : x \in L, k \geq 1\}$ of distributions, one for each $x \in L$ and security parameter k. We say that two ensembles of distributions E_1 and E_2 over L are indistinguishable with respect to a security parameter $k(|x|)$ if for any polynomial time machine (tester) T, all constants c, and all sufficiently long x it holds that

$$|\text{Prob}_{s \leftarrow E_1(x,k(|x|))}[T(s) = 1] - \text{Prob}_{s \leftarrow E_2(x,k(|x|))}[T(s) = 1]| \leq \frac{1}{k(|x|)^c},$$

where $s \leftarrow E(x,k)$ stands for s being sampled according to the distribution $E(x,k)$.

A remark about the uniformity of the tester. Note that we may define the tester to be nonuniform. In this case we would get a similar result for the nonuniform case. Namely, if there exists a one-way permutation that is robust against polynomial nonuniform attacks, then the zero-knowledge property (i.e., the indistinguishability between the distribution output by the simulation and the distribution of the proofs generated by the prover) would also hold for nonuniform testers.

We now define noninteractive zero-knowledge proof systems.

Definition 2.1 [5]. A *noninteractive zero-knowledge proof system* for a language L with security parameter k consists of a probabilistic proving algorithm P, a probabilistic polynomial time verification algorithm V, and a probabilistic polynomial time machine

(simulator) M with the following property. Let σ be a random string shared by the prover and verifier, and let $k(|x|)$ be a security parameter. Then:

1. **Completeness:** For any $x \in L$ and security parameter k,

$$\text{Prob}[V(x, \sigma, P(x, \sigma, k)) = \text{accept}] = 1,$$

   where the probability space is taken over the random coin tosses of P and V, and over the uniform and independent choice of the random bits in the shared random string σ.[2]
2. **Soundness:** For any $x \notin L$, and any (possibly cheating) prover P', and soundness error $\varepsilon(|x|)$,

$$\text{Prob}[V(x, \sigma, P'(x, \sigma, k), k) = \text{accept}] \leq \varepsilon(|x|),$$

   where the probability space is the same as in the completeness condition.
3. **Zero knowledge:** There exists a simulator M that, on input x and security parameter k, produces a distribution space, and the ensemble $\{M(x, k)\}_{x \in L}$ is indistinguishable from the ensemble $\{(\sigma, P(x, \sigma, k))\}_{x \in L}$ which is defined over the distribution space of a random uniformly chosen σ over $\{0, 1\}$ and the random choices of the prover P.

A remark about the soundness property. Usually, in the noninteractive zero-knowledge setting, the soundness property is stated in a stronger manner. Namely, it is required that a cheating prover succeeds with only small probability to prove *any* theorem of length n and not only the one theorem appearing in the input. Our protocol can be made resilient to such a demand simply by lowering the error probability by a factor of 2^{-n}. We adopt the soundness property that is usually used for the interactive model, so that a comparison of our protocol to the interactive model will be fair. In the interactive model there is no conceivable scenario in which the input should be chosen *after* the random tape of the verifier has been determined and shown to the prover. Note that the "standard" harsh soundness demand is an artifact of the noninteractive zero-knowledge model and not of any particular noninteractive zero-knowledge proof system.

In this paper we address the problem of circuit satisfiability. However, in order to simplify the presentation, we reduce this problem to a more restricted satisfiability problem: 3-SAT-5. The problem 3-SAT-5 is defined as follows:

Definition 2.2. The problem 3-SAT-5:

Input: A 3-CNF formula φ with each variable appearing in at most five clauses.
Question: Is φ satisfiable?

It is important that there exists a reduction from circuit satisfiability to 3-SAT-5 which is linear since we present a proof system for 3-SAT-5 with complexity $O(nk \log(n/\varepsilon))$,

[2] The definition of completeness may be relaxed to require only that the acceptance probability (when $x \in L$) be greater than $1 - \varepsilon(|x|)$ for some given completeness error ε. In our protocol we obtain perfect completeness as defined above.

where n is the size of the 3-SAT-5 formula, and we claim that this proof can serve for proving circuit satisfiability with complexity $O(nk \log(n/\varepsilon))$ where n is the size of the circuit. Indeed, circuit satisfiability can be easily reduced to 3-SAT using a linear reduction. Furthermore, the reduction from 3-SAT to 3-SAT-5 can also be done linearly, by creating a new variable for each copy of the original variable and adding clauses which force all copies of the same variable to have equal assignment. Thus, it is sufficient to present a noninteractive zero-knowledge proof system for 3-SAT-5.

3. The Compilation Technique of Feige et al. [11]

Here we formally define the *hidden bits model* of [11] and show how to use a one-way permutation to transform a zero-knowledge proof given in this model to a zero-knowledge proof in the standard (shared random bits) model. We later state our protocol in the hidden bits model. The introduction of the model and the proof that it is equivalent to the shared random string model were given in [11]. We follow the presentation suggested in [13].

3.1. *The Model*

In the hidden bits model the common random string R is selected uniformly, but only the prover can see its content. The prover sends the verifier a message Π and reveals a list of positions $I \subset \{1, 2, \ldots, \text{poly}(|x|)\}$ in the random string. That is, the verifier receives I and the correct values of the bits in the shared random string whose locations are mentioned in I. We denote by R_I the content of the hidden bits at the places pointed by I. Thus, R_I is a bit vector of length $|I|$, and if the sequence I includes the indices $i_1, i_2, \ldots, i_{|I|}$, then the jth entry in R_I contains R_{i_j}, i.e., the i_jth bit in the shared random string R. The verifier is not allowed to look at any other position in the shared random string R.

The soundness and completeness requirements remain the same as before. In order to show that a protocol in this model is zero-knowledge, one has to present a simulator that simulates the parts of the proof that the verifier can see. Namely, one has to present a simulator whose output distribution on x is computationally indistinguishable from the distribution of proofs $(x, R_I, P(x, R))$ for a randomly chosen R, and where $P(x, R) = (\Pi, I)$ is the distribution of proofs output by the prover on input x and the shared random tape R.

It is easier to design proof systems in this model. It was shown in [11] that given a one-way permutation, it is possible to compile a zero-knowledge proof system in the hidden bits model into a zero-knowledge proof system in the standard model. Actually, for the case of efficient provers, i.e., when the prover can be implemented efficiently when given some auxiliary help, the compilation requires a family of *trapdoor* permutations. We begin by presenting the compilation for the simple case where the prover is computationally unbounded, in which case we use one-way permutations. We then explain how to extend it to the case of efficient provers where we use trapdoor one-way permutations.

3.2. *The Compilation Procedure*

We give a brief exposition of the compiling procedure of [11]. Let (P, V) be a zero-knowledge proof system for a language L in the hidden bits model, let F be a one-way permutation, and let B be a hard-core bit of F as guaranteed by Goldreich and Levin [14] (it is shown in [14] that any one-way permutation can be transformed into a one-way permutation that has a hard-core bit, with a moderate increase in the domain size). Finally, let $m(n)$ represent the number of hidden bits required by the given proof system. We would like to construct a zero-knowledge noninteractive proof for L in the standard model.

In the standard (i.e., the shared random string) model the prover and verifier are given access to the input x and to a shared random string r. We require that r has length $k \times m(n)$, where k is the security parameter. This means that the overhead of this compilation in terms of the length of the random string is a multiplicative factor of k. The prover and verifier both regard the random string as being composed of $m(n)$ random strings of length k. In what follows, we denote these strings by s_i, $1 \le i \le m(n)$.

Intuitively, the strings s_i, $1 \le i \le m(n)$, will be regarded as hiding the bits $B(F^{-1}(s_i))$. Also, if the prover wishes to "show" the ith bit to the verifier, then he may send him the value $r_i = F^{-1}(s_i)$. The verifier can then check that indeed $F(r_i) = s_i$ and may also compute the hidden bit $B(r_i)$. By the properties of the hard-core predicate B, the verifier will not be able to guess the value of any hidden bit that is not explicitly shown to him by the prover.

The prover (P') acts as follows: He computes the bits $b_i = B(F^{-1}(s_i))$ for all $i = 1, 2, \ldots, m(n)$. He invokes the given prover P on the input x and hidden bits $b_1, b_2, \ldots, b_{m(n)}$ to get a pair (I, Π). He sends to the verifier V' the output (I, Π) of the prover P and also $r_i = F^{-1}(s_i)$ for each $i \in I$.

The verifier (V') verifies the proof as follows: V' checks that indeed $s_i = F(r_i)$ for each $i \in I$. Otherwise, V' rejects the proof. V' computes $b_i = B(r_i)$ and concatenates all b_i into b_I. Then V' invokes V on (x, b_I, I, π) and accepts only if V accepts.

Claim 3.1. *The proof system (P', V') constructed as above is a zero-knowledge noninteractive proof for L.*

Sketch of Proof. Note that since the s_i are uniformly distributed, and since F is a permutation, then $F^{-1}(s_i)$ is also uniformly distributed and since the hard-core bit B (as constructed in [14]) is unbiased, then the interpretation of the string $s_1, s_2, \ldots, s_{m(n)}$ as hiding the bits

$$B(F^{-1}(s_1)),\ B(F^{-1}(s_2)),\ \ldots,\ B(F^{-1}(s_{m(n)})$$

gives us completely random hidden bits. Also, note that the prover cannot influence the value of the bits in the positions he lets the verifier check, since any string s_i has only a single inverse r_i which then determines the bit $B(r_i)$. This guarantees that the soundness and completeness properties are kept.

For the zero-knowledge property, we note that we can compile an efficient simulation of P into an efficient simulation of P'. The new simulator invokes the original simulation to get (x, R_I, Π, I). It then chooses the strings s_i (forming the shared random string) as follows: For each $i_j \in I$, it selects a random r_j satisfying $B(r_j) = R_{i_j}$ and sets $s_{i_j} = F(r_j)$, and, for each $i \notin I$, it selects a random string $s_i \in \{0,1\}^k$. Then it outputs the created shared random string, i.e., $s_1, s_2, \ldots, s_{m(n)}$, and the proof which contains r_i for each $i \in I$ and also (I, Π) as created by the original simulator. In order to see that this simulation is indistinguishable from the distribution of proofs output by P', note that the original simulation is indistinguishable from the proofs generated by P, and that B is a hard-core for F. □

3.3. *Extension for Efficient Provers*

To extend this compilation for the case of efficient provers we use trapdoor one-way functions. In this case the prover chooses in the beginning a permutation with a trapdoor that allows him to invert it. He sends the description of the permutation to the verifier and then the protocol continues as above.[3]

For this case the completeness property is not affected. Namely, the honest prover succeeds with the same probability. Also, the construction of the new simulator from the old one is practically the same. The new simulator has to use the permutation given by the original simulation in the beginning of the proof (recall that the new simulator does not have to be able to invert this permutation). We still have to deal with the soundness property.

The problem here is that after the shared (hidden) random string is set, the prover may go over all 2^k possible permutations, and for each of them examine the different shared random tape that is implied by the permutation, and choose the one that is best for him. Note that although we would like to allow an efficient prover to play its part in the protocol, we do not want to trust the prover to be weak; we want the soundness property to be robust against powerful provers as well.

We cannot avoid preventing the prover from trying all possible permutations in the family, so instead we note that this advantage of the prover cannot increase the soundness error probability by more than a factor of 2^k. Namely, if the original soundness probability (in the hidden bits model) is ε, then the soundness probability of the compiled protocol (in the shared random string model) is $\varepsilon \cdot 2^k$. Therefore if one wishes to design a protocol for efficient provers with error probability ε, and one wishes to do the design in the hidden bits model, then the protocol should be designed with error probability $2^{-k} \cdot \varepsilon$. The fact that the error probability can only increase by 2^k (i.e., the number of permutations in the family) follows from simple counting arguments: For each specific function the error probability is at most ε and there are at most 2^k functions.

We further elaborate on the changes needed for efficient provers in Section 7.

[3] Actually, we have to take care of the prover indeed choosing a permutation in a predetermined family of permutations and not selecting a confusing representation which is not a permutation. This issue is dealt with in [3] and we do not discuss it here.

4. An Overview of the Protocol

In Section 5 we describe the protocol in full detail. The description there is a bottom-up description of how the proof system is built on top of the (hidden) random string. However, we believe that some overview discussion may help before getting into the details. We begin with describing *influence games* which is a basic idea underlying our protocol. Influence games, as an underlying idea of a secure construction, were first used independently in [17] and in [8]. Then we provide a short top-down description of the proof system, which may help going through the details of Section 5.

4.1. *Influence Games*

The protocol of [17], as well as ours, is based on *influence games*. In these games the prover and verifier have a series of strings with the following properties:

1. The prover can see what bits are hidden in the strings but the verifier cannot. (Note that these strings are not just the hidden bits, these are more complicated creatures as follows from the second property.)
2. There are two possible strings:
   - **Random strings:** These are strings of hidden bits that the prover can open and show their content to the verifier. The values of these bits are randomly chosen in the beginning of the proof and the prover cannot cheat, i.e., he can only open these bits to their predetermined value.
   - **Wild strings:** These are strings of hidden bits which the prover can open to whatever value he likes. Namely, the prover can open each bit of the string to either 0 or 1. (This is the "influence" of the game.)
3. The verifier cannot tell between random strings and wild strings.

During the protocol, there is an initial stage in which the prover and verifier work on building these "influence" strings (Parts 1–3 of our protocol—Section 5) and then there is a final stage in which they use them to prove (or verify, respectively) that the input is in the language (Part 4 in our protocol). Basically, the use of these strings gives the prover some "influence" in opening their values. His goal in this protocol is to open the strings such that some property regarding the opened bits hold. The protocol is designed so that when $x \in L$ the prover has enough influence to open the strings appropriately, whereas if $x \notin L$, then the prover does not have enough influence on the values of these bits and with high probability the bits he will open will not satisfy the desired property.

4.2. *A Top-Down Description of our Protocol*

We give a top-down description of the protocol based on the influence games described in the previous subsection. The input to the protocol is a 3-CNF formula φ with variables $x_1, x_2, \ldots, x_n$ and each variable appears in at most five clauses. Also, let τ be a satisfying assignment for φ (we describe the protocol for the honest prover). As explained in Section 4.1, in the beginning of the protocol the prover and verifier work on interpreting the given random string as strings for the influence game. Actually, they are going to build, for each variable x_i, a pair of strings (s_i^0, s_i^1) such that if $\tau(x_i) = \textbf{false}$, then the string s_i^0 is a wild string and the string s_i^1 is a random string, whereas if $\tau(x_i) = \textbf{true}$,

then the string s_i^1 is a wild string and the string s_i^0 is a random string. This is done in Parts 1–3 of the protocol. We now explain how the prover uses these strings to show that indeed τ satisfies φ.

For each of the five clauses in which x_i is involved we select the following substring. If x_i appears negated in a clause C_j, then we select a fifth of the string s_i^0 to represent the variable x_i in the clause C_j, whereas if x_i is positive (not negated) in the clause, then we select a fifth of the string s_i^1 to represent x_i in the clause C_j. The substrings are selected with no overlap, i.e., a character in a string is selected only for one clause. (Here we use the property that the variable appears in at most five clauses, and we selected only a fifth of the given string.) Note that the selection is made exactly so that if τ on x_i satisfies the clause, then the selected string is a wild string and the prover can open it any way he wants, whereas if the clause is not satisfied by the assignment τ to the variable x_i, then x_i is represented by a completely random string over $\{0, 1\}$. We stress that only the wild strings can be modified by the prover. The random strings are fixed (according to the content of the shared hidden string) and the prover cannot influence them.

After all strings for each clause have been selected, the prover opens the characters in the selected strings. Each character is opened to a bit (either a zero or a one) and thus we get for each clause three strings of bits. The verifier checks that for each clause the three strings of bits revealed by the prover sum up to zero. Namely, the bit-wise exclusive-or of the revealed strings is a string of zeros. If τ is indeed a satisfying assignment to φ, then the prover can easily pass this test. For each clause, one of the selected strings contains only wild characters and he can open this string to whatever bit string he needs. However, if τ does not satisfy φ, then there exists a clause for which all the selected strings are completely random and the prover will be caught with high probability. To be more specific, the strings are of length $O(\log(n/\varepsilon))$, and thus (if we set the constants properly) the probability that the prover will fail in a single clause test (that is not satisfied by τ) is less than $\Omega(\varepsilon/5n)$. Since the number of clauses is less than $5n$, the proof follows. (Of course, in the real proof the bits will not be "completely random" and we will have to be more careful in the analysis.)

Thus, given a pair of strings for each variable with the above characteristics, it is possible to check whether τ satisfies φ. However, we still have to explain how to build a pair of strings for each variable. We do that in the full description of the protocol in the next section.

5. The Protocol in Detail

We present a noninteractive zero-knowledge protocol for 3-SAT-5. As mentioned in Section 2, this implies the following theorem:

Theorem 1. *Circuit satisfiability can be proven by a noninteractive zero-knowledge proof system whose length is $O(nk \cdot \log(n/\varepsilon))$, where k denotes the security parameter, ε is the error probability, and n is the size of the circuit.*

As explained above, if we want efficient provers, then we loose a factor of k in the above and get that the length of the shared random string is $O(nk^2 \cdot \log(n/\varepsilon))$. In the

description we use the hidden bits model. See Section 3 for the details. We say that the prover "opens" a bit to the verifier meaning that the prover lets the verifier see this bit in the hidden random string. In the terminology of Section 3 this means that the prover includes the index of the bit in the set I.

The Protocol. Denote the input formula by φ and denote the number of variables in φ by n. Recall that in our model the prover and verifier share a random tape which only the prover can see and the prover specifies which bits the verifier will see too ("opens" bits for the verifier). The proof consists of the prover sending one message, together with the indices that the verifier may look at. The verifier, based on this message (the proof) and the bits he sees in the shared random (hidden) string, decides whether to accept the claim that the input formula is satisfiable.

In our proof system (as in other noninteractive zero-knowledge proof systems) the prover's message includes a statement or an interpretation of the structure of the shared random string in a way that several properties are satisfied if and only if the input (φ) is in the language (3-SAT-5).

Our proof system is a direct improvement over the protocol in [17] which in turn is inspired by the protocol of [11].

5.1. *Interpreting Pairs of Bits as Characters*

The prover and verifier first establish *characters* in the shared random hidden string. A character can have three possible values: 0, 1, and a wild character (WC). Characters are encoded by pairs of hidden bits in the following way:

00—the character 0,
10—the character WC,
01—the character WC,
11—the character 1.

Each pair of hidden bits is interpreted as a hidden character. We next set up a procedure by which the prover can "open" a character. Namely, a prover can show the verifier that a character (which consists of a pair of hidden bits) contains a certain value. Throughout the proof, the prover will open a character either to the value 1 or to the value 0. To prove that a character has value 0, the prover will open one of the two hidden bits and show that it is a 0. To show that a character is 1, the prover will open one of the hidden bits (in the pair) and show that it is a 1. Clearly, a wild character can be opened both ways, since both 0 and 1 can be opened in such a character. We make explicit use of the fact that the character WC can be opened by the prover both to the value 0 and to the value 1.

Whenever the prover has to open a character which is not a WC, he must choose which of the two bits to open. It is important for the zero-knowledge property that in all these cases the prover chooses between the two possibilities at random. If, for example, the prover always opens the first bit, then he reveals information about whether or not the opened characters are WC, since an opening of a second bit will imply that this is a WC character. In the rest of this description we assume that the decision of which bit to open is made at random.

To summarize our interpretation of the random tape so far, we have a stream of uniformly independently chosen characters of value 0, 1, and WC. The character 0 and the character 1 appear each with probability $1/4$ and the character WC appears with probability $1/2$. The prover can open the character 0 and 1 to their correct value, and he can open the character WC to both values 0 or 1 as he wishes.

5.2. *Sieving Good Blocks*

In the next step of the interpretation of the shared hidden random string we consider blocks of α consecutive characters (where α is a constant to be determined later (see Section 6.2.2). In fact, we consider the given stream of characters as a sequence of pairs of blocks. We call a pair of blocks *good* if one of the blocks in the pair contains only WC characters and the other block does not contain any WC character. For each variable x_i, $1 \leq i \leq n$, the prover initially considers $\beta \log(n/\varepsilon)$ pairs of blocks (for a constant β to be determined later (see Section 6.2.2), and the error probability ε) and lets only the good blocks prevail for the rest of the proof.

In the first part of the proof the prover completely opens each character in every block that is not good. These blocks are not used again in the proof. We stress that the prover not only opens these characters (as we defined an opening of a character by opening one of its hidden bits), but rather opens the two hidden bits in each of the characters, so that the verifier can check that the discarded blocks are indeed not good.

First Part of the Proof. *The prover opens all hidden bits in all pairs of blocks which are not good.*

The verifier verifies that all blocks that were opened were indeed not good. Otherwise he stops immediately and rejects. Also, the verifier checks for each variable x_i that the number of pairs of remaining blocks, denoted ℓ_i, i.e., the number of pairs of blocks that were claimed to be good for x_i (these are still hidden), is close to its expected value. If ℓ_i is not in the range $[t_1, t_2]$ to be specified below, then the verifier stops immediately and rejects.

We compute the expected value of ℓ_i and determine the bounds t_1 and t_2. The probability that a random pair of blocks is good is $2 \cdot (1/2)^\alpha \cdot (1/2)^\alpha$ (choose which member of the pair is the WC block, then it has probability $(1/2)^\alpha$ to consist only of WC characters, and the other block has probability $(1/2)^\alpha$ to contain no WC character inside). Therefore, the number of good blocks between the $\beta \log(n/\varepsilon)$ random blocks initially considered for the variable x_i is a random variable ℓ_i that has expected value $2^{-2\alpha+1} \cdot \beta \cdot \log(n/\varepsilon)$.

By the Chernoff bound, the deviation of ℓ_i from its expected value is big with low probability. Specifically,

$$\text{Prob}\left[\left|\ell_i - 2^{-2\alpha+1} \cdot \beta \log \frac{n}{\varepsilon}\right| \geq \gamma\beta \log \frac{n}{\varepsilon}\right] \leq 2 \cdot 2^{-\log(n/\varepsilon)\beta\cdot\gamma^2\cdot 2^{2\alpha-2}}. \tag{1}$$

The parameter γ is a constant which determines the tightness of the bound. We set its value later (see Section 6.2.2). Now set the deviation bounds that the verifier insists

on as

$$t_1 \stackrel{\text{def}}{=} (2^{-2\alpha+1} - \gamma)\beta \log \frac{n}{\varepsilon} \quad \text{and}$$
$$t_2 \stackrel{\text{def}}{=} (2^{-2\alpha+1} + \gamma)\beta \log \frac{n}{\varepsilon}.$$

By the Chernoff bound, if we set the constant γ according to

$$\gamma = \sqrt{\frac{3 \cdot 2^{-2\alpha+2}}{\beta}}, \tag{2}$$

then the random variable ℓ_i is between t_1 and t_2 with probability at least $1 - \varepsilon/4n$ for a random stream of characters.

After the good pairs of blocks have been selected (by discarding the bad ones), the prover and verifier leave only the first t_1 good blocks and discard the last $l_i - t_1$ good blocks. In this way we have a fixed number of blocks (i.e., t_1 blocks) for each of the variables. Note that when $x \in L$ this step can be done with high probability. We remark in Section 6.4 about what the prover may do if $x \in L$ and l_i is not between t_1 and t_2. When $x \notin L$ and the prover is trying to cheat, then, still with high probability, l_i is between t_1 and t_2. In this case the worst scenario is that the prover finds only t_1 good pairs, he claims that there are t_2 good pairs ($t_2 - t_1$ of them were actually bad), and by the end of the process he is left with t_1 blocks out of which $t_2 - t_1$ are actually bad pairs of blocks.

To summarize this step, for each variable x_i, the (honest) prover and verifier are left with t_1 good pairs of blocks, each consisting of one block of WC characters and one block containing no WC characters. A dishonest prover may claim that some bad pairs of blocks are good, but with probability at least $1 - \varepsilon/4n$ the number of pairs of blocks he can cheat on is at most $t_2 - t_1$.

Thus we end this step having a set of good pairs of blocks associated with each variable. Next, we begin to associate the shared random string (which is now a random stream of good pairs of blocks) with a satisfying assignment of the formula φ.

5.3. *Setting the Polarity of the Pairs of Blocks According to the Satisfying Assignment to φ*

Each pair of blocks contains a block of WC characters and a (random) block without WC characters. The order of the blocks in each pair is random. In the second part of the proof, the prover reorders these pairs according to some satisfying assignment to the variables in the formula φ. Let τ denote such an assignment. The prover reorders all of the block pairs corresponding to each variable x_i, so that if $\tau(x_i) = \textbf{false}$, then the first block in each block pair corresponding to x_i is set to be the WC block and if $\tau(x_i) = \textbf{true}$, then the second block in each block pair corresponding to x_i is set to be the WC block. The reordering of the pairs of blocks constitutes the second part of the proof.

Second Part of the Proof. *For each pair of blocks* (*obtained from the shared random tape*), *which was not dropped in the previous step of the proof, the prover specifies a bit. If a pair is assigned the bit* 0, *then its order is kept, whereas if a pair of blocks is assigned* 1, *then the order of the blocks in the pair is reversed for the rest of the proof.*

The verifier does not check anything, but only reverses the order of the blocks where necessary. Note that he is essentially rearranging pointers; he still does not know the values of the hidden characters.

To summarize this step, the *honest* prover now has pairs of blocks, each associated with a variable, such that each variable's pairs are ordered (polarized) in the same manner, corresponding to the variable's assignment.

In the following step we merely check that the pairs of blocks are indeed polarized consistently. This by itself does not guarantee that the polarization matches a satisfying assignment to the formula φ.

5.4. *Checking the Consistency of the Polarization*

We now describe a consistency test for the pairs of blocks associated with a variable x_i. This test is repeated for each of the variables. Recall that we have t_1 pairs of blocks which are supposed to be polarized. The test consists of many basic tests; each basic consistency test is performed on a couple of pairs. We have to specify what the basic test is and also which pairs are going to be tested against each other. We begin with the second.

We consider an expander graph with t_1 vertices (see, for example, [19] for a possible constructions of expanders). Each vertex corresponds to a pair of blocks, and the edges, as appearing in the expander, determine which pairs are going to be checked one against the other (through the basic test). In what follows, we denote the degree of the expander by d and the expansion rate by $\Delta = 1+\delta$. Namely, each subset A of the vertices of cardinality at most $|A| \leq t_1/2$ has at least $\Delta \cdot |A|$ neighbors, or at least $\delta \cdot |A|$ neighbors which are not in A. Actually, in order to get better parameters for the expanders (i.e., a smaller ratio of d/δ) and considering the requirements really needed by our constructions, it will suffice for us if the expansion property only holds for sets A of cardinality at least $t_1/10$.

Next we describe the basic consistency test as applied on two pairs of blocks. Let these two pairs be numbered j and k, and denote by (B_j^0, B_j^1) and (B_k^0, B_k^1) the blocks in these pairs. Recall that the requirement to make the basic test originates from the existence of an edge $e = (j, k)$ in the expander. We use $\alpha/2d$ specific characters of each of the four blocks involved in this test. Let e be the ith edge ($1 \leq i \leq d$) adjacent to vertex j representing the pair (B_j^1, B_j^2), then we use the characters $(\alpha \cdot (i-1))/2d + 1$, $(\alpha \cdot (i-1))/2d + 2, \ldots, (\alpha \cdot (i-1))/2d + \alpha/2d$ in B_j^0 and in B_j^1 to form the strings of characters $s_j^0(e)$ and $s_j^1(e)$ correspondingly. We do the same for the pair (B_k^0, B_k^1) to get $(s_k^0(e), s_k^1(e))$. So the character-strings which are used for the test specified by the edge $e = (j, k)$ are $(s_j^0(e), s_j^1(e))$ and $(s_k^0(e), s_k^1(e))$. Note that we never use the same character for two different tests. If the prover is honest and follows the protocol, then the blocks are good and well polarized, and then $s_k^t(e)$ and $s_j^t(e)$ contain only wild characters and $s_k^{1-t}(e)$ and $s_j^{1-t}(e)$ are random over $\{0, 1\}$, where t is 0 if $\tau(x_i) =$ **false** and 1 otherwise. In this case the prover can open the values of these characters such that

$$\begin{aligned} s_j^1(e) &= s_k^0(e) \quad \text{and} \\ s_j^0(e) &= s_k^1(e), \end{aligned}$$

since the prover can open the WC characters to whatever value he wants. We define this as the basic consistency test.

Third Part of the Proof. *The prover opens the first half of the characters in each block such that, for each edge $e = (j, k)$ in the expander, if e is the lth edge of vertex j and the mth edge of vertex k, and let $\hat{B}_j^t(i)$ denote the value that the prover opens in the block B_j^t for the ith character, then it should hold that the opened values satisfy*

$$\hat{B}_j^0\left(\frac{\alpha\cdot(l-1)}{2d}+i\right) = \hat{B}_k^1\left(\frac{\alpha\cdot(m-1)}{2d}+i\right) \quad \textit{and}$$
$$\hat{B}_j^1\left(\frac{\alpha\cdot(l-1)}{2d}+i\right) = \hat{B}_k^0\left(\frac{\alpha\cdot(m-1)}{2d}+i\right)$$

for all $1 \le i \le \alpha/2d$. Namely, all the basic consistency tests associated with all the expander edges hold.

The verifier verifies that all the characters which should have been opened were indeed opened, and that for each edge of each expander, the consistency test is satisfied.

To summarize, in this step we "lost" half of the characters which we cannot use in the following, but we gained some assurance that there is some consistency in the polarity of the pairs. We analyze the degree of this assurance in the analysis of the protocol (in Section 6).

For the next stage, we do the following concatenation process. For each of the variables, we concatenate all the first blocks in all its corresponding pairs to a single string, and concatenate all the second blocks in all its pairs to another string. So for each variable we have two strings, and if the prover is honest, then one of these strings is a string of WC characters only and the other string is a random string over $\{0, 1\}$. The length of each string is $t_1 \cdot \alpha/2$ since we are left with only half of the characters in each of the t_1 blocks.

5.5. *Last Step*: *Showing that τ Satisfies φ*

In this last step the prover uses the polarization of the pairs of strings, which should represent τ, to show that the formula φ is satisfiable. If the prover behaves according to the protocol, then it should hold that for each variable x_i, $1 \le i \le n$, the prover and verifier share a pair of (long) strings of (hidden) characters. One of the strings consists of WC characters only, and the other string is a random string over $\{0, 1\}$ which contains no wild characters. Also, the string of WC characters is the first in the pair if $\tau(x_i) =$ **false** and the second in the pair if $\tau(x_i) =$ **true**. Recall that the prover has the ability to open all the characters in the wild string either to 0 or to 1 as he wishes, but the other string is fixed.

For each x_i, we take the pair of long strings (composed of many pairs of blocks) and partition it into five pairs. One pair for each appearance of x_i in the formula. Each piece is still a pair of long strings that will be used to check the assignment to the clause. The details follow.

For each clause, we make a test. Loosely speaking, for each variable in the clause we take part of its string (of characters) so that if the variable satisfies the clause (i.e., it

is assigned **true** and appears positively in the clause or it is assigned **false** and appears with a negation in the clause), then the prover can completely control the opening of the string that corresponds to the variable. If τ assigns x_i a value that does not satisfy the clause, then ideally the prover can open the corresponding string in only one way which corresponds to a random string over $\{0, 1\}$ (in the full analysis, we must consider deviations from this ideal). After determining the three strings that correspond to the clause, the prover has to open the strings such that their bit-wise exclusive-or is equal to the all zero string. If the prover can control one of the strings (or, equivalently, τ makes one of the literals satisfy the clause), then he can easily pass this test. Otherwise, the prover has to open three random strings, and their exclusive-or is zero with small probability.

More formally, if clause C_i contains the lth appearance ($1 \leq l \leq 5$) of variable x_j and if the strings associated with x_j are (B_j^0, B_j^1) (each of length $t_1 \cdot \alpha/2$), then if the (negated) literal $\overline{x_j}$ appears in the clause then we select the bits $B_j^0(k)$ for

$$k = \frac{l-1}{5} \cdot \frac{t_1\alpha}{2} + 1, \frac{l-1}{5} \cdot \frac{t_1\alpha}{2} + 2, \ \ldots, \ \frac{l-1}{5} \cdot \frac{t_1\alpha}{2} + \frac{t_1\alpha}{10}$$

while if the (unnegated) literal x_j appears in the clause C_i then we select the bits $B_j^1(k)$ for

$$k = \frac{l-1}{5} \cdot \frac{t_1\alpha}{2} + 1, \frac{l-1}{5} \cdot \frac{t_1\alpha}{2} + 2, \ldots, \ \frac{l-1}{5} \cdot \frac{t_1\alpha}{2} + \frac{t_1\alpha}{10}.$$

Thus, for each clause, we select three strings of length $\alpha/10 \cdot t_1$. Intuitively, if the assignment τ satisfies one of the literals in the clause, then the string that corresponds to this literal contains only WC characters.

After selecting the three strings of the clause, the prover opens all the characters in all of these strings so that the bit-wise exclusive-or of the strings equals $0^{\alpha \cdot t_1/10}$.

In case the prover has more control over the opening of these characters (e.g., if more than one variable is assigned **true** in the clause and so more than one string is a wild string which can be opened arbitrarily), then the prover uses his degrees of freedom randomly. Namely, he opens all WC characters at random conditioned on the linear constraint that has to hold for the test. This random selection is crucial for the zero-knowledge property, since as we will see the simulator will open the characters randomly, conditioned only by the linear constraint posed by the proof. Also, recall that if the character is not WC, then the prover opens a randomly chosen bit in the pair.

Fourth Part of the Proof. *The prover opens all the remaining characters so that for each clause, its three corresponding strings* (*as explained above*) *have a bit-wise exclusive-or which equals* $0^{\alpha \cdot t_1/10}$.

The verifier checks that each clause indeed passes the test, and that all characters were indeed opened legitimately. In this case, he accepts. Otherwise, he rejects.

6. Analysis of the Protocol

6.1. *Completeness*

This is the easy part of the proof. It should be clear that the prover can perform all his tasks when the input formula φ is indeed satisfiable. He will fail only when the number of good blocks ℓ_i, for some variable x_i, is not in the range between t_1 and t_2. By the Chernoff inequality (see (1) and (2)), this happens with probability at most

$$n \cdot \frac{\varepsilon}{4n} < \varepsilon.$$

6.2. *Soundness*

We analyze the probability that the prover can produce a convincing proof (on a random string) for a nonsatisfiable formula φ. The prover convinces the verifier if:

1. For each variable x_i, the number of good blocks claimed by the prover is at least t_1 and at most t_2.
2. All polarity tests hold.
3. All clause tests hold.

Recall that ℓ_i denotes the actual number of good pairs of blocks in the shared hidden string, and not the number of pairs that the cheating prover might have chosen to keep. We assume that the prover always passes the first test. Actually, this is not the case, since if $\ell_i > t_2$, then the prover must fail the first test, because he cannot open enough bad pairs. However, our assumption may only increase the soundness error (i.e., the probability that the prover succeeds). It turns out that the prover can pass the other two tests with a very small probability, and this suffices for the soundness analysis.

We assume that if ℓ_i is not in the range between t_1 and t_2, then the prover always succeeds in convincing the verifier. Again, this is not correct, but this assumption can only increase the error. The case that one of the ℓ_i ($1 \le i \le n$) is not in the right range happens with probability at most

$$n \cdot \frac{\varepsilon}{4n} \le \frac{\varepsilon}{4}.$$

We continue the soundness analysis assuming that for all i, $1 \le i \le n$, it holds that $t_1 \le \ell_i \le t_2$. Conditioning on this event, we compute the probability that conditions 2 and 3 hold. We show that they hold with (conditional) probability at most $\varepsilon/2$. Thus, the probability that the prover can convince the verifier is at most ε. This holds for any possible $\varepsilon < 1/4$ chosen for the system.

Assume now that the ℓ_i's are all in the range between t_1 and t_2. We first note that in this conditional space one important property still holds as in the unconditional space. That is, the non-WC characters in the good block pairs are uniform and independent over $\{0, 1\}$. Namely, we note that the number of good pairs of blocks is independent of the specific content of the non-WC characters in the good pairs of blocks. This property is essential for the rest of the proof.

We assume a case that is best for the cheating prover. This can only increase the probability that the cheating prover succeeds in convincing the verifier. We assume that

the prover can claim that a lot of bad pairs of blocks are good pairs and keep them for the rest of the proof. The case in which he can keep the maximum number of bad blocks is when the actual number of good pairs, ℓ_i, is t_1. In this case the prover can claim that there are t_2 good pairs of blocks and leave $t_2 - t_1$ bad pairs for future use in the proof. The prover can also determine the location of these $t_2 - t_1$ pairs. Note that in any other case the prover has less control over the proof, since he can use and determine the locations of less pairs of blocks.

Furthermore, we allow the prover more freedom. We assume that, in all the bad pairs of blocks that the prover announces as good, all the characters are WC. Namely, the prover has complete control over these pairs and he can open each of the characters in these blocks to whatever value he wants. Note that in any other case the prover has less control since the value of some of the characters will be set to a specific bit instead of the prover being able to determine their values in a manner that best suits his goal.

Recall that after the good pairs of blocks are pointed out by the prover, we truncate the last pairs so that we are left with exactly t_1 pairs of (supposedly) good blocks for each of the variables x_i, $1 \leq i \leq n$. Again, we assume the best possible case for the prover in which the truncated pairs are all good pairs of blocks, while the $t_2 - t_1$ bad pairs of blocks still remain for the rest of the proof.

To summarize, we consider the worst case in which there are t_1 pairs of blocks left for the rest of the proof, out of which $t_2 - t_1$ pairs are bad. The bad pairs contain blocks with only WC characters, and the location of the $t_2 - t_1$ bad pairs in the list of t_1 pairs is determined by the prover. However, it is important to note that the blocks that do not contain WC characters contain strings that are uniformly and independently chosen in $\{0,1\}^\alpha$.

Next, we define an assignment τ to the variables of the formula φ. We shall show that if this assignment does not satisfy φ (as must be the case here since we assume that φ is not satisfiable), then conditions 2 and 3 hold with small probability. Consider the good pairs of blocks after being polarized in the second part of the (noninteractive) proof. For any variable x_i consider the majority of the polarization in the $t_1 - (t_2 - t_1)$ good-block pairs which correspond to x_i. Since the prover is not necessarily honest, some of these pairs may have their WC block as the first block and some of the pairs may have their WC block as the second block. We define $\tau(x_i)$ to be **false** if in the majority of the pairs the first block in the pair is the WC block, and we define $\tau(x_i)$ to be **true** if in the majority of the pairs the second block is the WC block (ties are broken arbitrarily).

Each variable has a degree of consistency with this assignment. Define the consistency of the variable x_i to be the fraction of the good pairs that have polarization equal to the majority polarization of the variable (by which $\tau(x_i)$ is determined). Clearly, each variable has consistency at least $1/2$. We are going to partition the analysis into two cases. One possible case is that one of the variables has consistency less than $8/9$, and then we show that the consistency test of the polarization (the third part of the proof) passes with low probability. The second possibility is that all variables are at least $8/9$ consistent. In this case we recall that τ cannot satisfy all clauses (since φ is not satisfiable) and we show that the prover can pass the test of a clause which is not satisfied by τ with low probability. We show that whichever of the above is valid, the prover succeeds in convincing the verifier with probability at most $\varepsilon/2$.

6.2.1. *A Probabilistic Argument*

For each of the above possibilities, we use the following method by which we prove that the prover fails with high probability. We define a randomized prover that chooses his cheating strategy at random, and compute the probability that the proof which the randomized prover produces passes the relevant tests. Next, we show that even if the prover chooses his best strategy (rather than selecting his strategy at random), the probability that he manages to convince the verifier still remains low.

We begin with an overview. The cheating prover has some freedom which emerges from his ability to claim that some bad blocks are good. He is going to use this freedom in order to change the order of the good blocks in the proof. For example, if the value of the third good block does not fit some test, then the prover can make it fourth by declaring a bad block (appearing before the third block) to be good. Thus, on top of his complete influence on the opening of bad blocks, the prover also gains some advantage by influencing the order of the good blocks. Note that (although we shall not use this fact) this freedom is given to him only once, and should allow him to pass all tests.

Next we note that the number of possible choices given to the prover can be upper bounded by some exponential (in the number of blocks) term which describes the number of possibilities to insert the bad blocks in between the good blocks. However, the probability of the verifier not detecting an error in the polarity tests or in the clause tests is exponentially small both in the number of good blocks and also in the *length* of each block. This difference gives us the edge in the soundness analysis, and a sufficiently large (still a constant) block-size would make the second expression (i.e., the probability to fail) kill the first expression (i.e., the freedom given to the prover). The formal details follow.

6.2.2. *Case I*: *There Exists an Inconsistent Variable*

Choose a variable x_i ($1 \leq i \leq n$) and assume that the consistency of x_i is lower than $8/9$. We calculate the probability that the prover has a winning strategy for the third part of the proof (the consistency test of the polarization). In the end of the (Case I) analysis, we multiply this probability by n (the number of variables) to compute an upper bound on the probability that there exists a variable with consistency less than $8/9$ and the prover still passes the consistency tests.

As mentioned, we begin by calculating the probability that the prover can pass the consistency test (third part of the proof) when he chooses his strategy at random. Recall that we are assuming a worst case scenario in which there are t_1 pairs of blocks for x_i out of which $t_2 - t_1$ pairs of blocks were declared good by the prover but contain, in fact, only WC characters.

In the polarization consistency test the prover has to open half of the characters in each block. The randomized prover opens all the non-WC characters to 0 or 1 as he must, and he opens all the WC characters in the best way, so that the test that they are engaged in is satisfied. (Recall that each character is engaged in at most one test.) However, the prover still has two things to choose. First, he can determine where to put the $t_2 - t_1$ bad pairs of blocks amongst the $\beta \log(n/\varepsilon)$ pairs. Second, he can decide which pairs are polarized inconsistently with the majority of the pairs. We let the randomized prover pick these at random. Namely, he picks at random $t_2 - t_1$ places for the bad

pairs between the $\beta \log(n/\varepsilon)$ pairs, and he picks $t_1/9$ (or more) good pairs that will be polarized inconsistently (with the polarity of the majority of the pairs).

What is the probability that the prover passes the consistency test? Recall that each basic consistency test is performed on two pairs. If the pairs are polarized in the same manner, then the test is easily passed by the prover opening the wild characters properly. Also, if anything is tested against a bad pair, then the prover easily passes the test since it has one of the pairs consisting of wild characters only. However, when two good pairs of opposite polarization are tested against each other, then each of the non-WC blocks in both pairs contributes a completely random string of $\alpha/2d$ bits and the two strings must be equal. This happens with probability exactly $2^{-\alpha/2d}$.

Note a delicate point in our claim that the strings compared are random. Indeed the contents of the strings were chosen uniformly at random, but it is known in advance which pairs are tested against each other and the prover may use his advantage in setting the *places* of the bad pairs or deciding *which pairs* are inconsistently polarized in order to set specific pairs against each other and enhance the probability of the good pairs to pass the test. Nevertheless, this is not the case here. We are considering now the randomized prover who chooses the places of the bad pairs and the good pairs that are inconsistently polarized at random, and thus the good pairs relate to one another in a completely independent and random manner.

Now we compute how many oppositely polarized good pairs are tested against each other. By our assumption, there are at least $(t_1 - (t_2 - t_1))/9$ good pairs which are not polarized consistently with the majority of the pairs. By the setting of the parameters α, γ (see below in this proof) we get that $(t_1 - (t_2 - t_1))/9 > t_1/10$. Therefore we have at least $t_1/10$ good pairs which are not polarized as the majority good pairs are polarized. By the expansion property of the expander graph we use, these pairs have at least $t_1 \cdot \delta/10$ neighbors that are either consistently polarized with the majority or are bad (not good) pairs. Since the number of bad pairs is at most $t_2 - t_1$, we get that at least $m \stackrel{\text{def}}{=} t_1 \cdot \delta/10 - (t_2 - t_1)$ tests are made between good pairs that are oppositely polarized. Since we do not recycle characters, all these tests are completely independent, and the test is passed with probability $2^{-\alpha m/2d}$.

Recall that

$$\begin{aligned} t_1 &= (2^{-2\alpha+1} - \gamma)\beta \log \frac{n}{\varepsilon} \quad \text{and} \\ t_2 &= (2^{-2\alpha+1} + \gamma)\beta \log \frac{n}{\varepsilon}. \end{aligned}$$

Thus,

$$m = \beta \log \frac{n}{\varepsilon} \left(\frac{\delta}{10} \cdot (2^{-2\alpha+1} - \gamma) - 2\gamma \right).$$

To summarize, the probability that the randomized prover passes this test is at most

$$2^{-\beta \log(n/\varepsilon)((\alpha\delta/20d)(2^{-2\alpha+1}-\gamma)-(\alpha/d)\gamma)}.$$

We now consider an arbitrary prover, not limited to a random strategy. The randomized prover chooses between its strategies at random. The choice is made between the $\binom{\beta \log(n/\varepsilon)}{t_2 - t_1}$ possibilities to fix the places of the bad pairs, and between choosing the

inconsistently polarized pairs. The number of options for this second choice can be bounded by 2^{t_1}. Also, the randomized prover passes the test with probability at most $2^{-\beta \log(n/\varepsilon)((\alpha\delta/20d)(2^{-2\alpha+1}-\gamma)-(\alpha/d)\gamma)}$. By simple counting arguments, if the prover uses his best strategy, then the probability, p, that the prover passes this test is at most

$$\begin{aligned} p &\leq \binom{\beta \log(n/\varepsilon)}{t_2 - t_1} \cdot 2^{t_1} \cdot 2^{-\beta \log(n/\varepsilon)((\alpha\delta/20d)(2^{-2\alpha+1}-\gamma)-(\alpha/d)\gamma)} \\ &\leq 2^{\beta \log(n/\varepsilon)(H(2\gamma)+2^{-2\alpha+1}-\gamma-(\alpha\delta/20d)(2^{-2\alpha+1}-\gamma)+(\alpha/d)\gamma)}, \end{aligned}$$

where H is the entropy (in base 2). For the values of γ and α that we shall choose it holds that $H(2\gamma) \leq 4.5\alpha\gamma$ and thus,

$$p \leq 2^{\beta \log(n/\varepsilon)(5\alpha\gamma+2^{-2\alpha+1}(1-\alpha\delta/20d))}.$$

Setting $\alpha = 21d/\delta$ (for example, for the expanders suggested by [19], we get $\alpha \leq 155$),

$$p \leq 2^{\beta \log(n/\varepsilon)(5\alpha\gamma-(1/20)2^{-2\alpha+1})}.$$

We also set $\gamma = 2^{-2\alpha+1}/125\alpha$ and $\beta = 3 \cdot (125\alpha)^2 \cdot 2^{2\alpha}$ (note that this setting satisfies the requirement posed in (2)) and we obtain

$$p \leq 2^{-2\log(n/\varepsilon)} < \left(\frac{\varepsilon}{n}\right)^2.$$

So the prover passes the consistency test of a specific x_i with probability at most $(\varepsilon/n)^2$. Thus, the probability that there exists an inconsistent variable yet the prover can still pass all the consistency tests is at most $\varepsilon^2/n < \varepsilon/4$.

6.2.3. *Case II*: *All Variables Are Almost Consistent*

Assume that the consistency of all the variables is at least $8/9$. Again, we consider the randomized prover that acts as before. That is, the prover selects at random the places of the $t_2 - t_1$ bad pairs of blocks in between the $\beta \log(n/\varepsilon)$ blocks, and selects at random which good pairs will have an inconsistent polarization. In all other senses (i.e., opening WC characters) the prover always makes his best choice in order to pass all tests.

Recall that we have defined an assignment τ that corresponds to the majority of the polarities in each variable. Since the input formula is not satisfiable (we are analyzing the soundness of the protocol) there exists a clause that τ does not satisfy. We compute the probability that the prover can pass the clause test for this specific clause. In the end of the (Case II) analysis, we multiply this probability by the number of clauses in order to get an upper bound on the probability that the prover passes all the clause tests while there exists a clause that is not satisfied by τ.

Fix a specific clause and let the literals in this clause be y, z, and w. (Each of them is either a variable or a negation of a variable.) Each variable contributes a string of characters s_x, s_y, and s_z of length $\alpha/10$. We get only $\alpha/10$ characters from each block since we used $\alpha/2$ characters for the consistency check and since we use the rest of the $\alpha/2$ characters for five different clause tests. Recall that we never use a character twice.

Since τ assigns each of the literals the value **false** and the literals are $8/9$ consistent, then we can deduce the following on each one of the strings.

Each of the strings s_x, s_y, and s_z is composed of some random characters and some WC characters. We check how many WC characters can be there. By the guarantee on the consistency, we know that at least $8/9$ of the good pairs contribute random blocks to each of the strings. However, there are also the bad blocks which contain only WC characters. So adding both of these numbers, we get that the number of WC characters is at most $\frac{1}{9}(t_1-(t_2-t_1))+t_2-t_1$, and by the setting of the parameters α and γ, we get that this is at most $t_1/5$. This bound implies that in each string there are at least $\frac{4}{5}\cdot t_1\cdot\alpha/10$ random characters and at most $\frac{1}{5}\cdot t_1\cdot\alpha/10$ WC characters.

Note again that the claim on randomness is based on the prover selecting the places of the bad pairs and the inconsistently polarized good pairs at random.

Recall that for the clause test the prover must open all characters such that the bit-wise exclusive or of the strings s_x, s_y, and s_z equals $0^{t_1\cdot\alpha/10}$. What is the probability that the randomized prover passes this test? For the prover, the best possible arrangement of the WC characters in the three strings s_x, s_y, and s_z is when their indices do not overlap, and then the prover has control over $\frac{3}{5}\cdot t_1\cdot\alpha/10$ characters. Still, all the rest of the characters are completely random and sum up to 0 with probability $1/2$. Since all these characters are completely random and independent, the probability that the randomized prover passes this test is at most

$$2^{-2\alpha t_1/50}.$$

As before, we consider an arbitrary prover. The number of possible strategies from which the randomized prover has chosen at random is bounded by

$$\left(\binom{\beta\log(n/\varepsilon)}{2\gamma\beta\log(n/\varepsilon)}\cdot 2^{t_1}\right)^3$$

(as in Case (I) but for three variables). So if the prover chooses the best strategy instead of picking a strategy at random, the probability, p, that he passes the test satisfies:

$$\begin{aligned} p &\le 2^{3\beta\log(n/\varepsilon)H(2\gamma)+3t_1-2\alpha t_1/50} \\ &= 2^{\beta\log(n/\varepsilon)\cdot(3H(2\gamma)-\gamma(3-2\alpha/50)+2^{-2\alpha+1}(3-2\alpha/50))}. \end{aligned}$$

Again, using the assignments to α, β, and γ, we get that the probability that the prover passes this test even when he chooses his best strategy is at most $(\varepsilon/n)^2$.

Now, we remember that this calculation applies to a specific clause. Since (in the worst case) the prover may choose the unsatisfied clause, and since there are at most $5n$ clauses (in 3-SAT-5 formula with n variables), then the probability that the prover will pass all the clause tests although there exists a clause that τ does not satisfy is at most $5\varepsilon^2/n \le \varepsilon/4$.

6.2.4. *Combining Both Cases*

In both cases the prover is caught with high probability. If the variables are not polarized consistently, then he is caught at the consistency test with probability at least $1-\varepsilon/4$ and if he sets the polarization of the variables almost consistently, then he is caught in

at least one of the clause tests with probability at least $1 - \varepsilon/4$. Combining both, we conclude that the prover is caught with probability at least $1-\varepsilon/2$. Also, our analysis was conditioned on the event that the conditions given in the Chernoff inequality hold (see (1)) and by the setting of the parameters this happens with probability $\varepsilon/4$. Therefore, the overall probability that the prover manages to convince the verifier on an input φ which is not satisfiable is at most ε.

6.3. *Zero Knowledge*

In order to prove the zero-knowledge property of the protocol, we have to show that there exists a probabilistic polynomial time simulator M which on input $\varphi \in$ 3-SAT-5 outputs a distribution on the message, revealed indices and contents of the hidden bit string at these indices, and whose output distribution is computationally indistinguishable from the distribution output by the (honest) prover on a uniformly chosen hidden bit string.

The simulator begins by producing a genuine random string which will help him simulate the hidden bit string. Next, the simulator follows the prover strategy as far as it can. Namely, it produces characters, gathers the characters into blocks, and reveals indices and contents of all the bad blocks. At this stage (before the second part of the proof), the simulator replaces all the characters in the good blocks by WC characters. The simulator does that by picking at random between the pairs "01" or "10" and replacing the original "hidden" bits in these places by the new bits. Note that in the rest of the proof only one bit in each character is opened, and the content of this bit as well as the order of this bit in the pair will be completely random as in the original proof, except for the linear constraints checked by the verifier which hold both in the proof and in the simulation.

After substituting the characters, the simulator polarizes the blocks randomly. Namely, for each remaining block, a bit is chosen uniformly and independently and these bits are sent in the second part of the proof. The remaining pairs of blocks are subsequently used in the order specified by this random polarization.

Next, the simulator passes all the required tests by opening the relevant characters appropriately. Note that each test is made of two or three characters of which at least one must be a WC character and the rest can be completely random. The opening of the WC character in the original proof is done so that it satisfies the linear constraint posed by the proof. In the simulation the simulator produces the same distribution on the opened characters. Namely, all the opened characters are uniformly chosen in $\{0, 1\}$ given that they have to satisfy the relevant test. (Note that it is not correct to always open a pair to 0–1 when the exclusive-or of these bits has to equal 1, since in the original proof the pair 0–1 has the same probability as 1–0.) Since no character is used twice, the distribution of the proof output by the simulation (on satisfiable formulas) is identical to the distribution of the real proof.

To summarize, since the bits opened by the simulation are completely random (uniformly chosen) conditioned on the linear constraints posed by the verification process, and since this is also the case for the real prover, on a uniformly chosen hidden bit string, the output of the simulation is exactly identical to the original proof. Note, however, that we cannot claim from *perfect* zero knowledge since the compilation procedure of

[11] (see Section 3) imposes a distribution that is only computationally indistinguishable from the original proof.

6.4. *A Remark on Perfect Completeness*

It is desirable for a proof system to have a perfect completeness property. Namely, that if the input formula is indeed satisfiable, then the prover will always be able to prove that this is the case. However, in our protocol even if the input formula φ is satisfiable, the prover may still not be able to prove it since the conditions stated in the Chernoff bound do not always hold for the shared random string of the proof.

To fix the protocol to have perfect completeness, we allow the prover to show that this is the case (by opening all the characters of the shared random string and showing that the number of good blocks is not within the desired bound). In this case the verifier accepts (although he gets no statement about the input formula). Using this augmented protocol, we gain perfect completeness and the soundness analysis remains unchanged since we have assumed that in this rare case, the prover always succeeds in convincing the verifier.

6.5. *Complexity*

Our proof uses $O(\log(n/\varepsilon))$ hidden bits for each variable, i.e., $O(k\log(n/\varepsilon))$ random bits for each variable. Thus the overall number of hidden bits we use is $O(n\log(n/\varepsilon))$ and the length of the shared random tape we use is $O(kn\log(n/\varepsilon))$.

7. Efficient Provers

The compilation procedures of [11] and [3] can be made to work for polynomial-time bounded provers with access to a witness for the NP problem. However, a few changes have to be made which affect the cryptographic assumption we use. We give a further exposition of these techniques in this section.

We assume the existence of a *family of certified trapdoor permutations*. Loosely speaking, a family $\mathcal{F}$ of trapdoor permutations is a set of permutations $\{f_i : \{0,1\}^k \to \{0,1\}^k\}_{i\in I}$ such that: $f_i(x)$ can be efficiently computed given (i,x), it is hard to compute $f_i^{-1}(y)$ given (i,y), for each i there exists a *trapdoor* $t(i)$ such that it is possible to compute efficiently $f_i^{-1}(y)$ given $(i,y,t(i))$, and there exists an efficient algorithm that given the security parameter k, samples $i \in I \cap \{0,1\}^k$ uniformly together with $t(i)$, i.e., there is an efficient sampler that samples $(i,t(i))$ such that i is uniformly chosen in $I \cap \{0,1\}^k$.

The additional "certification" property is that given i, it is possible to verify efficiently that $i \in I$, i.e., $f_i \in \mathcal{F}$. Specifically, if f_i is in $\mathcal{F}$, then we know that f_i is a permutation. Actually, the need for the certification is removed by [3].

Our proof system for an efficient prover involves an additional preliminary step. Note that the only use we made of the power of the prover is in inverting the one-way permutation while interpreting the shared random string in the compilation procedure. This enabled the shared random string to be interpreted as hidden bits only shown to the prover.

In order to remove this need for powerful computation, we let the prover choose at random $(i, t(i))$ such that $i \in I \cap \{0,1\}^k$ and the whole proof is carried out with the one-way permutation f_i. At the beginning of the proof, the prover states the index i he is going to use and the verifier verifies that indeed $i \in I$. Note that it is important to verify this since f_i must be a permutation for the soundness of the proof system. After that, the proof system is the same and the efficient prover can invert f_i (using the trapdoor $t(i)$) whenever needed.

The completeness and the zero-knowledge analysis remain the same, but the soundness property has to be computed again, since the prover gains a new power: he can interpret the shared random string in $|I \cap \{0,1\}^k|$ different manners (according to the choice of $f_i \in \mathcal{F}$) and perhaps he can choose f_i that most helps him to cheat on a nonsatisfiable input formula φ. Note that although the honest prover can be run efficiently, we would like to defend ourselves against computationally unbounded cheating provers.

We treat the soundness analysis in the same manner as we did in the original soundness analysis. Namely, first we analyze what happens when the prover really chooses uniformly $f_i \in \mathcal{F}$ and then we compute the advantage he may get when picking the best possible $f_i \in \mathcal{F}$. So when f_i is picked at random, the probability that the prover succeeds in cheating the verifier is the same as computed in Section 6.2. Therefore, the probability that the prover succeeds in cheating when he can pick $f_i \in \mathcal{F}$ that best serves his purpose is at most

$$\varepsilon \cdot |I \cap \{0,1\}^k| \leq \varepsilon \cdot 2^k,$$

as needed.

To summarize, in order to run the compilation procedure with an efficient prover, we use a family $\mathcal{F}$ of certified trapdoor permutations, we add a preliminary stage in which the prover chooses and states a member $f_i \in \mathcal{F}$, and then the prover and verifier interpret the shared random string according to the permutation f_i in the rest of the proof. The soundness probability suffers a loss of a factor of 2^k.

7.1. *Eliminating the Certification Requirement*

The need for certification in this setting was first noted by Bellare and Yung [3]. They also suggested a way to eliminate this requirement. Loosely speaking, they replace the requirement for efficient verification that f_i is a permutation by a proof (of the prover) that f_i is "close" to a permutation. The idea of the proof is that the prover has to invert random strings in $\{0,1\}^k$. In our setting, after the prover selects $i \in I \cap \{0,1\}^k$, he uses a predetermined part of the shared random string to demonstrate the permutation property of f_i. This part of the random string is partitioned into blocks of k bits and, for each block $u \in \{0,1\}^k$, the prover specifies $f_i^{-1}(u)$, i.e., the inverse of the block u under the mapping f_i.

If a δ fraction of the strings in $\{0,1\}^k$ do not have preimages and the prover tries to invert $(\log(1/\omega)) \cdot \delta^{-1}$ strings in $\{0,1\}^k$, then he is caught with probability at least $1-\omega$. Denote by l the number of hidden bits used in the proof, i.e., $l = O(kn \cdot \log(n/\varepsilon))$. We select $\delta \stackrel{\text{def}}{=} 1/l \cdot \log(n/\varepsilon)$ to be the fraction of bad strings in $\{0,1\}^k$ which we allow, and $\omega = \varepsilon/n$ to be the degree of reassurance. Therefore, we have to use additional $O(\log(1/\omega) \cdot \delta^{-1})$ (which is $O(l)$) tests to make sure that with probability at most ε/n

it holds that f_i violates this property and the prover is not caught. Note that we only multiply the length of the random string by a constant, so the complexity is not changed.

We first note that since there are at most 2^k possible permutations that the prover can choose, the test fails one of them with probability at least $1 - 2^k \cdot \varepsilon$. However, this loss of a factor of 2^k in the soundness error probability was already taken into account so we do not count it again. So now assume that we are dealing with a function l_i for which at most δ fraction of the points in $\{0,1\}^k$ have two inverses. In this case the prover gains an additional power in the proof: the ability to open the committed bits that have two inverses in both possible ways. We call these bits *bad*. As before, we let the prover choose the opening of the bad committed bits at random and we then conclude that the prover can use this to raise his probability to cheat the verifier by a factor of at most 2^b where b is a random variable that represents the number of bad committed bits between the l bits used for the original proof.

If the number of bad committed bits $b \leq 3\log(n/\varepsilon)$, then we are done, since our soundness analysis is robust to the prover increasing his number of possibilities by a factor of $(n/\varepsilon)^3$. So we compute the probability that the number of bad bits b does not exceed $3\log(n/\varepsilon)$. The expected number of bad bits is $l \cdot \delta = \log(n/\varepsilon)$. By the Chernoff inequality, we get

$$\begin{aligned} \text{Prob}\left[b \geq 3\log\frac{n}{\varepsilon}\right] &\leq \text{Prob}\left[|b - E[b]| \geq 2\log\frac{n}{\varepsilon}\right] \\ &\leq \frac{\varepsilon}{n}. \end{aligned}$$

It was shown in [3] that this proof is zero knowledge. It should also be clear that if $\varphi \in$ 3-SAT-5 and the (honest) prover indeed selects a permutation, then he cannot fail.

Summing up, we can use the technique of [3] in order to relax the cryptographic assumption from the existence of a family of *certified* trapdoor permutations to the existence of a general family of trapdoor permutations. The cost is at most a constant multiplicative factor in the length of the shared random tape.

8. Open Questions

The result in this paper follows earlier work on making noninteractive zero-knowledge protocols more efficient. However, there is no known lower bound on the complexity of a noninteractive proof for NP-Hard languages. It is a most intriguing open question whether nontrivial lower bounds can be proven in this case.

Acknowledgments

We thank Oded Goldreich and the anonymous referees for many useful remarks on earlier versions of this manuscript, which greatly improved its readability.

References

[1] D. Beaver, J. Feigenbaum, J. Kilian, and P. Rogaway. Security with Low Communication Overhead. In A. J. Menezes and S. A. Vanstone, editors, *Advances in Cryptology—CRYPTO '90 Proceedings*, pages 62–76. Lecture Notes in Computer Science, volume 537. Springer-Verlag, Berlin, 1990.

[2] M. Bellare and S. Goldwasser. New Paradigms for Digital Signatures and Message Authentication Based on Non-Interactive Zero-Knowledge Proofs. In J. Feigenbaum, editor, *Advances in Cryptology—CRYPTO '89 Proceeding, Lecture Notes in Computer Science*, volume 435, pages 194–211. Springer-Verlag, Berlin, 1989.

[3] M. Bellare and M. Yung. Certifying Cryptographic Tools: The Case of Trapdoor Permutations. In *Advances in Cryptology—CRYPTO '92 Proceeding*. Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1992.

[4] M. Blum, A. De Santis, S. Micali, and G. Persiano. Noninteractive Zero Knowledge. *SIAM J. Comput*., 20(6), 1991.

[5] M. Blum, P. Feldman, and S. Micali. Non-Interactive Zero Knowledge and Its Applications. In *Proc*. 20*th ACM Symp*. *on Theory of Computing*, pages 103–112, 1988.

[6] J. Boyar and R. Peralta. Efficient Zero-Knowledge Proofs of Circuit Satisfiability. Technical Report 1, ISSN No. 09033920, Institut for Matematik og Datalogi, Odense Universitet, 1994.

[7] I. Damgård. Non-Interactive Circuit-Based Proofs and Non-Interactive Perfect Zero Knowledge with Preprocessing. In *Advances in Cryptology—Eurocrypt '92 Proceeding*. Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1992.

[8] A. De Santis, G. Di Crescenzo, G. Persiano, and M. Yung. On Monotone Formula Closure of SZK. In *Proc*. 35*th Symp*. *on Foundations of Computer Science*, pages 454–465, 1994.

[9] A. De Santis, S. Micali, and G. Persiano. Non-Interactive Zero-Knowledge with Preprocessing. In S. Goldwasser, editor, *Advances in Cryptology—CRYPTO '88 Proceeding*, pages 27–35. Lecture Notes in Computer Science, volume 403. Springer-Verlag, Berlin, 1988.

[10] U. Feige. Personal communication.

[11] U. Feige, D. Lapidot, and A. Shamir. Multiple Non-Interactive Zero-Knowledge Proofs Based on a Single Random String. In *Proc*. 31*st Ann*. *Symp*. *on Foundations of Computer Science*, pages 308–317, 1990.

[12] A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Advances in Cryptology—CRYPTO '86 Proceeding*, pages 186–189. Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1986.

[13] O. Goldreich. Foundation of Cryptography—Fragments of a Book. Available from the *Electronic Colloquium on Computational Complexity* (*ECCC*) `http://www.eccc.uni-trier.de/eccc/`, February 1995. See Addendum, February 1996.

[14] O. Goldreich and L.A. Levin. A Hard Core Predicate for All One Way Functions. In *Proc*. 21*st ACM Symp*. *on Theory of Computing*, pages 25–32, 1989.

[15] S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM J. Comput*., 18(1):186–208, 1989.

[16] J. Kilian. A Note on Efficient Zero-Knowledge Proofs and Arguments (extended abstract). *Proc*. 24*th Ann*. *ACM Symp*. *on Theory of Computing*, pages 723–732, 1992.

[17] J. Kilian. On the Complexity of Bounded-Interaction and Non-Interactive Zero-Knowledge Proofs. In *Proc*. 35*th Ann*. *Symp*. *on Foundations of Computer Science*, 1994.

[18] J. Kilian, S. Micali, and R. Ostrovsky. Minimum Resource Zero-Knowledge Proofs. In *Proc*. 30*th Ann*. *Symp*. *on Foundations of Computer Science*, pages 474–479, 1989.

[19] A. Lubotsky, R. Phillips, and P. Sarnak. Ramanujan Graphs. *Combinatorica* 8(3):261–277, 1988.

[20] M. Naor and M. Yung. Public Key Cryptosystems Provably Secure Against Chosen Ciphertext Attacks. In *Proc*. 22*nd Ann*. *ACM Symp*. *on Theory of Computing*, pages 427–437, 1990.