

# Interactive PCP

Yael Tauman Kalai\*

Georgia Institute of Technology  
yael@csail.mit.edu

Ran Raz†

Weizmann Institute of Science  
ran.raz@weizmann.ac.il

## Abstract

A central line of research in the area of PCPs is devoted to constructing short PCPs. In this paper, we show that if we allow an additional interactive verification phase, with very low communication complexity, then for some NP languages, one can construct PCPs that are significantly shorter than the known PCPs (without the additional interactive phase) for these languages. We give many cryptographic applications and motivations for our results and for the study of the new model in general.

More specifically, we study a new model of proofs: **interactive-PCP**. Roughly speaking, an interactive-PCP (say, for the membership  $x \in L$ ) is a proof-string that can be verified by reading only one of its bits, with the help of an interactive-proof with very small communication complexity. We show that for membership in some NP languages  $L$ , there are interactive-PCPs that are significantly shorter than the known (non-interactive) PCPs for these languages.

Our main result is that for any constant depth Boolean formula  $\Phi(z_1, \dots, z_k)$  of size  $n$  (over the gates  $\wedge, \vee, \oplus, \neg$ ), a prover, Alice, can publish a proof-string for the satisfiability of  $\Phi$ , where the size of the proof-string is  $\text{poly}(k)$ . Later on, any user who wishes to verify the published proof-string needs to interact with Alice via a short interactive protocol of communication complexity  $\text{poly}(\log n)$ , while accessing the proof-string at a single location.

Note that the size of the published proof-string is  $\text{poly}(k)$ , rather than  $\text{poly}(n)$ , i.e., the size is polynomial in the size of the witness, rather than polynomial in the size of the instance. This compares to the known (non-interactive) PCPs that are of size polynomial in the size of the instance. By reductions, this result extends to many other central NP languages (e.g., SAT,  $k$ -clique, Vertex-Cover, etc.).

More generally, we show that the satisfiability of  $\bigwedge_{i=1}^n [\Phi_i(z_1, \dots, z_k) = 0]$ , where each  $\Phi_i(z_1, \dots, z_k)$  is an arithmetic formula of size  $n$  (say, over  $\mathbb{GF}[2]$ ) that computes a polynomial of degree  $d$ , can be proved by a published proof-string of size  $\text{poly}(k, d)$ . Later on, any user who wishes to verify the published proof-string needs to interact with the prover via an interactive protocol of communication complexity  $\text{poly}(d, \log n)$ , while accessing the proof-string at a single location.

We give many applications and motivations for our results and for the study of the notion of interactive PCP in general. In particular, we have the following applications:

1. **Succinct zero knowledge proofs:** We show that any interactive PCP, with certain properties, can be converted into a zero-knowledge interactive proof. We use this to construct zero-knowledge proofs of communication complexity polynomial in the size of the witness, rather than polynomial in the size of the instance, for many NP languages.
2. **Succinct probabilistically checkable arguments:** In a subsequent paper, we study the new notion of *probabilistically checkable argument*, and show that any interactive PCP,

---

\*Supported in part by NSF CyberTrust grant CNS-0430450. Part of this work was done when the author visited the Weizmann Institute.

†Supported by Binational Science Foundation (BSF), Israel Science Foundation (ISF) and Minerva Foundation.

with certain properties, translates into a probabilistically checkable argument [KR07]. We use this to construct probabilistically checkable arguments of size polynomial in the size of the witness, rather than polynomial in the size of the instance, for many NP languages.

3. **Commit-Reveal schemes:** We show that Alice can commit to a string  $w$  of  $k$  bits, by a message of size  $\text{poly}(k)$ , and later on, for any predicate  $\Phi$  of size  $n$ , whose satisfiability can be proved by an efficient enough interactive PCP with certain properties, Alice can prove the statement  $\Phi(w) = 1$ , by a zero-knowledge interactive proof with communication complexity  $\text{poly}(\log n)$ . (Surprisingly, the communication complexity may be significantly smaller than  $k$  and  $n$ ).

## 1 Introduction

Different interpretations and views of the notion of *proof* have played a central role in the development of complexity theory. Many of the most exciting ideas in complexity theory were originated by defining and studying new models of proofs. Three of the most successful models that were suggested are: Interactive Proofs [GMR85, BM88, LFKN90, S92], where the proof is interactive, Probabilistically Checkable Proofs (PCP) [FGLSS91, AS92, ALMSS92], where the verifier is only allowed to read a small number of symbols of the proof, and Multi-prover Interactive Proofs [BGKW88, BFL90], where the verifier interacts with several (say, two) different provers that are not allowed to communicate between them.

A two provers interactive proof can be viewed as a proof where the verifier is given access to two independent interactive proofs. In light of the great success of this model, it seems very interesting to study the case where the verifier is given access to two independent proofs of a different nature. For example, in [R05] the case where the verifier is given access to both, a quantum proof on one hand, and a classical PCP on the other hand, was studied.

Since interactive proofs and PCPs are among the most exciting models of proofs that were suggested, it seems very interesting to ask what happens when the two models are combined. In this paper, we study the new model where the verifier is given access to both, a PCP on one hand, and an interactive proof on the other hand. We think of this model also as an Interactively Verifiable PCP (or, in short, Interactive PCP), that is, a PCP that is verified by an interaction between a prover and the verifier. We show that in some cases this model has advantages over both PCPs and interactive proofs (given separately). More specifically, we show that the membership in many NP languages can be proved by a combination of a PCP and an interactive proof, where the PCP is much shorter than the best PCPs known for these languages, and the interactive proof is of a much lower communication complexity than the best interactive proofs known for these languages.

For example, one of our main results shows that for any constant depth Boolean formula  $\Phi(z_1, \dots, z_k)$  of size  $n$  (over the gates  $\wedge, \vee, \oplus, \neg$ ), a prover, Alice, can publish a proof-string (PCP) for the satisfiability of  $\Phi$ , where the size of the proof-string is  $\text{poly}(k)$  (rather than  $\text{poly}(n)$ , i.e., the size of the proof-string is polynomial in the size of the witness, rather than polynomial in the size of the instance). Later on, any user who wishes to verify the published proof-string needs to interact with Alice via an interactive protocol of communication complexity  $\text{poly}(\log n)$ , while accessing the proof-string at a single location.

By reductions, the result extends to many other central NP languages, e.g., SAT,  $k$ -clique, Vertex-Cover, etc. Moreover, a subsequent theorem of Goldwasser, Kalai, and Rothblum [GKR07], in the context of computation delegation, improves the above mentioned result so that it holds for any Boolean formula  $\Phi(z_1, \dots, z_k)$  of size  $n$  (rather than only for constant depth formulas). More

generally, they show that the same result holds for any Boolean circuit  $\Phi(z_1, \dots, z_k)$  of size  $n$  and depth  $d$ , where now the size of the proof-string is  $\text{poly}(k, d)$ , and the interactive verification phase is of communication complexity  $\text{poly}(d, \log n)$ .

We give many motivations and applications for these results, and for the study of the new model in general. Most of these applications are general reductions that convert any interactive PCP, with certain properties, into another object (i.e., the application). The motivations and applications are described in details in Subsection 1.4.

## 1.1 Interactive PCP

In this paper, we study a new model of proofs: **interactive PCP**, and show that for membership in some NP languages  $L$ , there are interactive-PCPs that are significantly shorter than the known (non-interactive) PCPs for these languages.

An interactive PCP (say, for the membership  $x \in L$ ) is a combination of a PCP and a short interactive proof. Roughly speaking, an interactive PCP is a proof that can be verified by reading only a small number of its bits, with the help of a short interactive proof.

More precisely, let  $L$  be an NP language, defined by  $L = \{x : \exists w \text{ s.t. } (x, w) \in R_L\}$ . Let  $p, q, l, c, s$  be parameters as follows:  $p, q, l$  are integers and  $c, s$  are reals, s.t.  $0 \leq s < c \leq 1$ . (Informally,  $p$  is the **size of the PCP**,  $q$  is the **number of queries** allowed to the PCP,  $l$  is the **communication complexity** of the interactive proof,  $c$  is the **completeness** parameter and  $s$  is the **soundness** parameter). We think of the parameters  $p, q, l, c, s$  as functions of the instance size  $n$ . An interactive PCP with parameters  $(p, q, l, c, s)$  for membership in  $L$  is an interactive protocol between an (efficient<sup>1</sup>) prover  $P$  and an (efficient) verifier  $V$ , as follows:

We assume that both the prover and the verifier know  $L$  and get as input an instance  $x$  of size  $n$ , and the prover gets an additional input  $w$  (supposed to be a witness for the membership  $x \in L$ ). In the first round of the protocol, the prover generates a string  $\pi$  of  $p$  bits. (We think of  $\pi$  as an encoding of the witness  $w$ ). The verifier is still not allowed to access  $\pi$ . The prover and the verifier then apply an interactive protocol, where the total number of bits communicated is  $l$ . During the protocol, the verifier is allowed to access at most  $q$  bits of the string  $\pi$ . After the interaction, the verifier decides whether to accept or reject the statement  $x \in L$ . We require the following (standard) completeness and soundness properties:

There exists an (efficient) verifier  $V$  such that:

1. Completeness: There exists an (efficient) prover  $P$ , such that: for every  $x \in L$  and any witness  $w$  (given to the prover  $P$  as an input), if  $(x, w) \in R_L$  then the verifier accepts with probability at least  $c$ .
2. Soundness: For any  $x \notin L$  and any (not necessarily efficient) prover  $\tilde{P}$ , and any  $w$  (given to the prover  $\tilde{P}$  as an input), the verifier accepts with probability at most  $s$ .

For the formal definition of interactive PCP, see Section 2.

Note that in the above definition we allow  $\pi$  to depend on  $L, x$  and  $w$ . However, in our results we use  $\pi$  that depends only on  $w$ , and is of size polynomial in the size of  $w$ . We hence think of  $\pi$  as an encoding of the witness  $w$  (and this encoding will always be efficient in our results). The fact that  $\pi$  depends only on  $w$  (and not on  $x$ ) is important for many of our applications.

Note also that the notion of interactive PCP is very related to the notion of multi-prover interactive proof [BGKW88]. For example, an interactive PCP with  $q = 1$  can be viewed as a two

---

<sup>1</sup>We could also consider a model with a not necessarily efficient prover.

provers interactive proof, where the interaction with the first prover is of only one round and is of question size  $\log p$  and answer size 1, and the interaction with the second prover is of communication complexity  $l$ , (and where both provers are efficient).

## 1.2 Our Results

We show that the membership in some NP languages, with small witness size, can be proved by *short* interactive PCPs with  $q = 1$ . We have two main results.

1. Let  $\Phi(z_1, \dots, z_k)$  be a constant depth Boolean formula of size  $n$  (over the gates  $\wedge, \vee, \oplus, \neg$ ). For any constant  $\epsilon > 0$ , the satisfiability of  $\Phi$  can be proved by an interactive PCP with the following parameters. Size of the PCP:  $p = \text{poly}(k)$ . Number of queries:  $q = 1$ . Communication complexity of the interactive proof:  $l = \text{poly}(\log n)$ . Completeness:  $c = 1 - \epsilon$ . Soundness:  $s = 1/2 + \epsilon$ .

Moreover, the string  $\pi$  (generated by the prover in the first round of the protocol) depends only on the witness  $w_1, \dots, w_k$ , and not on the instance  $\Phi$ .

2. Let  $\Phi_1(z_1, \dots, z_k), \dots, \Phi_n(z_1, \dots, z_k)$  be arithmetic formulas of size  $n$  (say, over  $\mathbb{GF}[2]$ ) that compute polynomials of degree  $d$ . For any constant  $\epsilon > 0$ , the satisfiability of the formula

$$\bigwedge_{i=1}^n [\Phi_i(z_1, \dots, z_k) = 0]$$

can be proved by an interactive PCP with the following parameters. Size of the PCP:  $p = \text{poly}(k, d)$ . Number of queries:  $q = 1$ . Communication complexity of the interactive proof:  $l = \text{poly}(d, \log n)$ . Completeness:  $c = 1$ . Soundness:  $s = 1/2 + \epsilon$ .

Moreover, the string  $\pi$  (generated by the prover in the first round of the protocol) depends only on the witness  $w_1, \dots, w_k$  (and on the parameter  $d$ ), and not on  $\Phi_1, \dots, \Phi_n$ .

The result works over any other finite field.

In both results, we could actually take  $\epsilon$  to be poly-logarithmically small. Also, the constant  $1/2$ , in the soundness parameter of both results, appears only because the string  $\pi$  is a string of bits. We could actually take  $\pi$  to be a string of symbols in  $\{1, \dots, 2^k\}$  and obtain soundness  $2^{-k} + \epsilon$ .

An additional property of our constructions is that we can assume that the verifier queries the string  $\pi$  before the interaction with the prover starts. This is the case, because the queries to  $\pi$  are non-adaptive (i.e., they do not depend on the values returned on previous queries) and do not depend on the interaction with the prover.

Note that many of the central NP languages can be reduced to the satisfiability of a constant depth formula, without increasing the witness size (e.g., SAT,  $k$ -clique, Vertex-Cover, etc.). We hence obtain short interactive PCPs for many other NP languages. Moreover, many NP languages can be reduced to the satisfiability of a formula of the form  $\bigwedge_{i=1}^n [\Phi_i(z_1, \dots, z_k) = 0]$  (without increasing the witness size), where  $\Phi_1, \dots, \Phi_n$  are arithmetic formulas of small degree. In these cases, by the second result, perfect completeness can be obtained.

## 1.3 Subsequent Result

In a subsequent work [GKR07], Goldwasser et al. improved our results as follows.

Let  $\Phi(z_1, \dots, z_k)$  be any Boolean circuit of size  $n$  and depth  $d$ . For any constant  $\epsilon > 0$ , the satisfiability of  $\Phi$  can be proved by an interactive PCP with the following parameters. Size of the

PCP:  $p = \text{poly}(k, d)$ . Number of queries:  $q = 1$ . Communication complexity of the interactive proof:  $l = \text{poly}(d, \log n)$ . Completeness:  $c = 1$ . Soundness:  $s = 1/2 + \epsilon$ .

Moreover, the string  $\pi$  (generated by the prover in the first round of the protocol) depends only on the witness  $w_1, \dots, w_k$ , and not on the instance  $\Phi$ .

Thus, in particular, Goldwasser et al. improve our first result so that it holds for any formula (rather than for a constant-depth formula) and with perfect completeness. Consequently, their result improves all the applications.

## 1.4 Motivations and Applications

Below we give several applications and motivations for our results, and for the study of the model of interactive PCP in general. In most of these applications what we actually have is a general reduction that converts any interactive PCP, with certain properties, into another object (i.e., the application). For simplicity, we concentrate on the applications of our first main result and the improved result given in [GKR07].

### 1.4.1 Motivation: Succinct PCPs with interaction

The PCP theorem states that the satisfiability of a formula  $\Phi(z_1, \dots, z_k)$  of size  $n$  can be proved by a proof of size  $\text{poly}(n)$  that can be verified by reading only a constant number of its bits [BFL90, FGLSS91, AS92, ALMSS92].

A central line of research in the area of PCPs is devoted to constructing short PCPs. An extremely interesting question is: Do there exist PCPs of size polynomial in the size of the witness, rather than polynomial in the size of the instance (think of the instance as significantly larger than the witness)? For example, does the satisfiability of a formula  $\Phi(z_1, \dots, z_k)$  of size  $n$  can be proved by a PCP of size  $\text{poly}(k)$ , rather than  $\text{poly}(n)$  (think of  $n$  as significantly larger than  $k$ )? A positive answer for this question would have important applications in complexity theory and cryptography (see for example [HN06]). However, a very interesting recent result by Fortnow and Santhanam shows that this is very unlikely, as it implies that  $NP \subseteq coNP/poly$  [FS07].

Our main results imply that for any constant depth Boolean formula  $\Phi(z_1, \dots, z_k)$  of size  $n$ , Alice can publish on the internet a “succinct” proof for the satisfiability of  $\Phi$ , where the size of the proof is  $\text{poly}(k)$  (rather than  $\text{poly}(n)$ , i.e., the size of the proof is polynomial in the size of the witness, rather than polynomial in the size of the instance). Later on, any user who wishes to verify the published proof needs to interact with Alice via an interactive protocol of communication complexity  $\text{poly}(\log n)$ , while accessing the published proof at a single location. By reductions, the result extends to many other central NP languages (e.g., SAT,  $k$ -clique, Vertex-Cover, etc.).

Using the above mentioned improvement of [GKR07] the same holds for any formula  $\Phi(z_1, \dots, z_k)$  of size  $n$ , (rather than a constant depth formula). Moreover, the result holds for any Boolean circuit  $\Phi(z_1, \dots, z_k)$  of size  $n$  and depth  $d$ , where now the size of the published proof is  $\text{poly}(k, d)$ , and the interactive verification protocol is of communication complexity  $\text{poly}(d, \log n)$ .

### 1.4.2 Application: Succinct probabilistically checkable arguments

In a very recent subsequent work [KR07], we give the following application of interactive PCPs. We study the new notion of *probabilistically checkable argument* (PCA) and we give a general way to construct a PCA from an interactive PCP. We use this to construct short PCAs for some languages in NP.

A probabilistically checkable argument (PCA) is a relaxation of the notion of probabilistically checkable proof (PCP). It is defined analogously to PCP, except that the soundness property is required to hold only *computationally*, rather than information theoretically. We consider the model where each verifier is associated with a public key, and each PCA is verifier-dependent, that is, it depends on the verifier’s public key. (The key does not need to be certified, and we can assume that the verifier simply publishes it on his web-page). We show that for membership in some languages  $L$ , there are PCAs that are significantly shorter than the known PCPs for these languages.

More precisely, our reduction in [KR07], combined with the above mentioned result of [GKR07] (that was obtained roughly at the same time as [KR07]), gives the following result: the satisfiability of a Boolean formula  $\Phi(z_1, \dots, z_k)$  of size  $n$  can be proved by a PCA of size  $\text{poly}(k)$ . That is, the size of the PCA is polynomial in the size of the witness (as opposed to known PCPs, that are of size polynomial in the size of the instance). The number of queries to the PCA is poly-logarithmic in  $n$ . As before, by reductions, the result extends to many other central NP languages (e.g., SAT,  $k$ -clique, Vertex-Cover, etc.). Moreover, the result holds for any Boolean circuit  $\Phi(z_1, \dots, z_k)$  of size  $n$  and depth  $d$ , where now the size of the PCA is  $\text{poly}(k, d)$  and the number of queries to the PCA is  $\text{poly}(d, \log n)$ .

The soundness property relies on exponential hardness assumptions.

### 1.4.3 Application: Succinct zero-knowledge proofs

The notion of *zero-knowledge proof*, first introduced by Goldwasser, Micali and Rackoff [GMR85], has become one of the central notions of modern cryptography. Goldreich, Micali and Wigderson showed that for any language  $L \in \text{NP}$ , the membership  $x \in L$  can be proved by an interactive zero-knowledge proof of polynomial communication complexity [GMW86]. An extremely interesting open problem in cryptography is: Can we significantly reduce the communication complexity of zero-knowledge protocols? Kilian and Micali, independently, showed that for any language  $L \in \text{NP}$ , the membership  $x \in L$  can be proved by a succinct interactive zero-knowledge *argument* of poly-logarithmic communication complexity [K92, M94]. Note, however, that the succinct zero-knowledge protocols of [K92, M94] are *arguments*, rather than *proofs*, that is, their soundness property holds *computationally*. These works left open the problem of constructing “short” zero-knowledge *proofs* for NP.

As an application of our results we show that the satisfiability of a constant depth Boolean formula  $\Phi(z_1, \dots, z_k)$  of size  $n$  (over the gates  $\wedge, \vee, \oplus, \neg$ ) can be proved by an interactive zero-knowledge proof of communication complexity  $\text{poly}(k)$  (rather than  $\text{poly}(n)$ ). That is, we obtain zero-knowledge proofs of communication complexity polynomial in the size of the witness, rather than polynomial in the size of the instance. As before, the result extends to many other central NP languages.

We note that a similar result, for the case of constant depth formulas, was proved independently (and roughly at the same time) by Ishai, Kushilevitz, Ostrovsky and Sahai [IKOS07], using different methods.

Once again, using the above mentioned improvement of [GKR07] the same holds for any formula  $\Phi(z_1, \dots, z_k)$  of size  $n$  (rather than a constant depth formula). Moreover, the result holds for any Boolean circuit  $\Phi(z_1, \dots, z_k)$  of size  $n$  and depth  $d$ , where now the communication complexity of the zero-knowledge proof is  $\text{poly}(k, d)$ .

We note that for this application we do not use the full power of interactive-PCP, and use mainly the fact that the interactive phase is of very low communication complexity. The results

are proved by a general reduction that converts any interactive PCP, with certain properties, into a zero-knowledge interactive proof. The reduction is obtained using standard and well known methods in cryptography (see for example [BGGHKMR88]).

#### 1.4.4 Application: Succinct Commit-Reveal schemes

The zero-knowledge proofs that we construct have an additional property that makes them very useful for many applications. They consist of two phases: The first phase is non-interactive, and depends only on the witness  $w = (w_1, \dots, w_k)$  (and is independent of the instance  $\Phi$ ). In this phase the prover sends to the verifier a certain (non-interactive) commitment to her witness at hand. The second phase is interactive and is very short. It is of communication complexity poly-logarithmic in  $n$ . In this phase the prover and verifier engage in a zero-knowledge proof that indeed the string that the prover committed to (in the first phase) is a valid witness for  $\Phi$ .

This additional property has the following immediate application: Given a string of bits,  $w = (w_1, \dots, w_k)$ , a user, Alice, can publish a commitment to  $w$ . The commitment is of size  $\text{poly}(t, k)$ , where  $t$  is the security parameter, and is non-interactive. Later on, for any constant depth Boolean formula  $\Phi(z_1, \dots, z_k)$  of size  $n$  (over the gates  $\wedge, \vee, \oplus, \neg$ ), such that  $\Phi(w) = 1$ , Alice can prove the statement  $\Phi(w) = 1$ , by an interactive zero-knowledge *proof* (rather than argument) with communication complexity  $\text{poly}(t, \log n)$ . Verifying this proof requires accessing the published commitment in only  $\text{poly}(t)$  locations. In other words, after publishing the commitment to  $w$ , Alice can prove the statement  $\Phi(w) = 1$ , by a zero-knowledge proof with communication complexity poly-logarithmic in  $n$ .

Note that the proof of the statement  $\Phi(w) = 1$  is of communication complexity poly-logarithmic in  $n$ , and may be significantly shorter than the length of  $w$ . This is interesting even if we discard the requirement of the proof being zero-knowledge.

Once again, using the above mentioned improvement of [GKR07] the same holds for any formula  $\Phi(z_1, \dots, z_k)$  of size  $n$  (rather than a constant depth formula). Moreover, the result holds for any Boolean circuit  $\Phi(z_1, \dots, z_k)$  of size  $n$  and depth  $d$ , where now the commitment is of size  $\text{poly}(t, k, d)$ , and the communication complexity of the zero-knowledge proof phase is  $\text{poly}(d, t, \log n)$ .

#### 1.4.5 Application: How to commit to a formula

Below, we give several examples for situations where the commit-reveal protocol of Section 1.4.4 may be useful. Many of these motivations and applications are taken from [KR06] (where succinct non-interactive argument systems with similar properties were given). Note, however, that the situation here is completely different than the one in [KR06], as here we give in the proof phase *interactive proofs* and there we gave *non-interactive arguments*. Nevertheless, many of the motivations and applications are similar.

For simplicity, we describe all these applications using the above mentioned improved result of [GKR07]. That is, we present the applications for any Boolean formula<sup>2</sup>, rather than only for constant depth formulas. We note that one could get results for formulas (rather than for constant depth formulas) also by using our second main result. These results, however, are not as strong as the ones obtained by using the improved result of [GKR07].

In all the applications that we describe below, except for the first one, the main idea is that the string  $w$  can itself be a description of a formula  $\Lambda$ . Thus, the commitment that Alice publishes is just

---

<sup>2</sup>Using [GKR07], one can also obtain similar results for circuits, rather than formulas, where the depth of the circuit is also taken into account. For simplicity, we present the applications here only for formulas.

a commitment to the formula  $\Lambda$ . We then take the formula  $\Phi$  to be a (*universal*) formula that runs the formula  $\Lambda$  on  $N$  different inputs,  $x_1, \dots, x_N$ , and checks that the  $N$  outputs are  $z_1, \dots, z_N$ . Thus, Alice publishes a commitment to  $\Lambda$ , and later on proves the statement  $\bigwedge_{i=1}^N [\Lambda(x_i) = z_i]$ , by a very short interactive zero-knowledge proof. The commitment is of size  $\text{poly}(t, |\Lambda|)$  (where  $t$  is the security parameter), and the communication complexity of the zero-knowledge proof is  $\text{poly}(t, \log |\Lambda|, \log N)$ . Note that the communication complexity is logarithmic in both  $|\Lambda|$  and  $N$ .

The main drawback of our protocol, in these contexts, is that it works only for Boolean formulas, and not for general Boolean circuits (with unbounded depth).

1. One of the main tasks of cryptography is to protect honest parties from malicious parties in interactive protocols. Assume that in an interaction between Alice and Bob, Alice is supposed to follow a certain protocol  $\Phi$ . That is, on an input  $x$ , she is supposed to output  $\Phi(x, w)$ , where  $w$  is her secret key. How can Bob make sure that Alice really follows the protocol  $\Phi$ ? A standard solution, effective for many applications, is to add a commitment phase and a proof phase as follows: Before the interactive protocol starts, Alice is required to commit to her secret key  $w$ . After the interactive protocol ends, Alice is required to prove that she actually acted according to  $\Phi$ , that is, on inputs  $x_1, \dots, x_N$ , her outputs were  $\Phi(x_1, w), \dots, \Phi(x_N, w)$ . In other words, Alice is required to prove the  $N$  statements  $\Phi(x_i, w) = z_i$ . Typically, we want the proof to be zero-knowledge, since Alice doesn't want to reveal her secret key.

Thus, Alice has to prove in zero-knowledge the statement  $\bigwedge_{i=1}^N [\Phi(x_i, w) = z_i]$ . The only known way to do this is by a proof of length  $N \cdot q$ , where  $q$  is the size of proof needed for proving a single statement of the form  $\Phi(x_i, w) = z_i$ . Note that  $N \cdot q$  may be significantly larger than the total size of all other messages communicated between Alice and Bob.

Our results imply that if  $\Phi$  is a Boolean formula, there is a much more efficient way. Alice will commit to her secret key  $w$ . Then, Alice can prove to Bob the statement  $\bigwedge_{i=1}^N [\Phi(x_i, w) = z_i]$ , by a zero-knowledge proof of size  $\text{poly}(t, \log |\Phi|, \log N)$ , (where  $t$  is the security parameter). That is, the proof is of size polylogarithmic in  $|\Phi|$  and  $N$ .

2. Alice claims that she found a short formula for factoring integers, but of course she doesn't want to reveal it. Bob sends Alice  $N$  integers  $x_1, \dots, x_N$  and indeed Alice manages to factor all of them correctly. But how can Bob be convinced that Alice really applied her formula, and not, say, her quantum computer? We suggest that Alice commits to her formula  $\Lambda$ , and then proves that she actually used her formula  $\Lambda$  to factor  $x_1, \dots, x_N$ . The commitment is of size  $\text{poly}(t, |\Lambda|)$  (where  $t$  is the security parameter), and the communication complexity of the zero-knowledge proof is  $\text{poly}(t, \log |\Lambda|, \log N)$ .
3. We want to run a chess contest between formulas. Obviously, the parties don't want to reveal their formulas (e.g., because they don't want their rivals to plan their next moves according to it). Of course we can just ask the parties to send their next move at each step. But how can we make sure that the parties actually use their formulas, and don't have teams of grandmasters working for them? We suggest that each party commits to her formula  $\Lambda$  before the contest starts. After the contest ends, each party will prove that she actually played according to the formula  $\Lambda$  that she committed to. As before the commitment is of size  $\text{poly}(t, |\Lambda|)$  and the communication complexity of the zero-knowledge proof is  $\text{poly}(t, \log |\Lambda|, \log N)$  (where  $N$  is the number of moves played).
4. Assume that both Alice and Bob have access to a very large database  $[(x_1, z_1), \dots, (x_N, z_N)]$ . They face a learning problem: Their goal is to learn a small Boolean formula  $\Lambda$  that explains



the database. That is, the goal is to learn  $\Lambda$  such that  $\Lambda(x_1) = z_1, \dots, \Lambda(x_N) = z_N$ . Alice claims that she managed to learn such a formula  $\Lambda$ , but she doesn't want to reveal it. It follows from our result that Alice can prove to Bob the existence of such a  $\Lambda$  by a zero-knowledge proof with communication complexity  $\text{poly}(t, |\Lambda|, \log N)$ , where  $t$  is the security parameter.

## 1.5 Techniques

Our proofs combine many techniques that were previously used in constructing PCPs, and in computational complexity in general, together with some new techniques. Our main technical contribution is a new *sum-check* procedure, which is, in many cases, more efficient than the standard one. More precisely, the standard sum-check procedure requires a prover that runs in time polynomial in the size of the space (on which the sum-check is performed). Our new procedure enables to perform sum-check, with a polynomial-time prover, in many cases where the space is of super-polynomial size. We hope that this new procedure may be interesting in its own right and may have additional applications in constructions of PCPs and interactive proofs.

The first result is proved by a reduction to the second result. This is done by approximating a constant depth formula by a (family of) polynomials of degree  $d = \text{poly}(\log n)$  (a well known method in complexity theory, first used by Razborov and Smolensky [R87, S87] for proving lower bounds for constant depth formulas). It is well known that the approximation can be done by a relatively small family of polynomials. That is, the approximation can be done using a relatively small number of random bits (see for example the survey paper of [B93]).

Suppose that we have a small family of polynomials of degree  $d = \text{poly}(\log n)$  that approximate the constant depth formula. After the prover generates the string  $\pi$ , the verifier chooses randomly a polynomial  $\Phi$  from the family, and the prover is required to prove that  $\Phi(w_1, \dots, w_k) = 1$  for the witness  $w = (w_1, \dots, w_k)$  encoded by  $\pi$ . This proves that the constant depth formula is satisfied by the witness  $w$ . We loose the perfect completeness because the low degree polynomials only approximate the constant depth formula and are not equivalent to it.

For the proof of the second result we use methods that were previously used for constructing PCPs and interactive proofs, together with some new techniques. The proof has two parts: The first part shows the needed result but with  $q$  larger than 1, that is, more than one query to the PCP. The second part shows that in any interactive PCP, the number of queries,  $q$ , can be reduced to 1, using a short additional interaction with the prover.

For the proof of the first part, we take  $\pi$  to be the *low degree extension* of the witness  $w$ . The verifier checks that  $\pi$  is indeed a low degree polynomial using a *low degree test*, as is frequently done in constructions of PCPs (e.g., [BFL90, AS92, ALMSS92]). Given an arithmetic formula  $\Phi$  of size  $n$  and degree  $d$ , the verifier can verify that  $\Phi(w_1, \dots, w_k) = 0$  by an interactive *sum-check procedure*, as is frequently done in constructions of PCPs and interactive proofs (e.g., [LFKN90, S92]). However, we need to apply the sum-check procedure on a space of size  $> k^d$ , which is, in most interesting cases, super-polynomial in  $n$ . This seems to require a prover that runs in super-polynomial time. Nevertheless, we show that we can use our new sum-check procedure, that can be performed by an efficient prover, even when the space is of super-polynomial size.

Note that the prover needs to prove that  $\Phi_i(w_1, \dots, w_k) = 0$  for every polynomial  $\Phi_i \in \{\Phi_1, \dots, \Phi_n\}$ . Since this would require too much communication, we take  $\Phi$  to be a (pseudo-random) linear combination of  $\Phi_1, \dots, \Phi_n$  (using any linear error correcting code). The combination is chosen by the verifier, and the prover is only required to prove that  $\Phi(w_1, \dots, w_k) = 0$ .

As mentioned above, the second part of the proof of the second result is a general theorem that shows that in any interactive PCP the number of queries  $q$  can be reduced to 1 (using some

additional interaction with the prover). This is done as follows. First, we can assume w.l.o.g. that all the queries to  $\pi$  are made after the interactive protocol ends. This is because rather than querying  $\pi$ , the verifier can simply ask the prover for the answers, and after the interactive protocol ends make the actual queries and compare the answers. Second, we can assume w.l.o.g. that the string  $\pi$ , generated by the (honest) prover, is a multivariate polynomial of low degree. Otherwise, we just ask the prover to generate the low degree extension of  $\pi$ , rather than  $\pi$  itself.

We can now apply a method that is based on methods that are frequently used in constructions of PCPs (e.g., [FL92, ALMSS92, DFKRS99, R05]). If the verifier wants to query  $\pi$  in  $q$  points, the verifier takes a curve  $\gamma$  of degree  $q+1$  that passes through all these points and an additional random point. If  $\pi$  is a low degree polynomial, the restriction of  $\pi$  to  $\gamma$  is also a low degree polynomial. The verifier asks the prover to send this low degree polynomial, and verifies the answer by checking it in a single point, using a single query to  $\pi$ . The verifier still needs to check that  $\pi$  is indeed a low degree polynomial. This is done, once again, using a low degree test. The verifier asks the prover for the restriction of  $\pi$  to a low dimensional subspace  $\nu$ , and verifies the answer by checking it in a single point, using a single query to  $\pi$ . This, however, requires an additional query to  $\pi$ , while we only allow one query altogether. We hence need to combine the two tests. That is, we need to combine the low degree test and the test that the prover's answer for the curve  $\gamma$  is correct. To do this, the verifier actually asks the prover to send the restriction of  $\pi$  to a manifold spanned by both  $\gamma$  and  $\nu$ . The verifier verifies the answer by checking it in a single point, using a single query to  $\pi$ .

This shows that the verifier can make only one query to  $\pi$ . However, the point queried in  $\pi$  contains a field element and not a single bit. To reduce the answer size to a single bit, the prover is required to generate in  $\pi$  the error correcting code of each field element, rather than the element itself. The verifier can now verify an element by querying only one bit in its error correcting code.

## 2 Definition of Interactive PCP

Let  $L$  be any NP language defined by  $L = \{x : \exists w \text{ s.t. } (x, w) \in \mathcal{R}_L\}$ . Let  $p, q, \ell, c, s$  be parameters that satisfy the following: The parameters  $p, q, \ell : \mathbb{N} \rightarrow \mathbb{N}$  are integers, and the parameters  $c, s : \mathbb{N} \rightarrow [0, 1]$  are reals, such that for every  $n \in \mathbb{N}$ ,  $0 \leq s(n) < c(n) \leq 1$ .

**Definition 1.** A pair  $(P, V)$  of probabilistic polynomial time interactive Turing machines is an **interactive PCP** for  $L$  with parameters  $(p, q, \ell, c, s)$ , if for every  $(x, w) \in \mathcal{R}_L$  the prover  $P(x, w)$  generates a bit string  $\pi$  of size at most  $p(n)$  (where  $n = |x|$ ), such that the following properties are satisfied.

- **Completeness:** For every  $(x, w) \in \mathcal{R}_L$ ,

$$\Pr[(P(x, w), V^\pi(x)) = 1] \geq c(n)$$

(where  $n = |x|$ , and the probability is over the random coin tosses of  $P$  and  $V$ ).

- **Soundness:** For every  $x \notin L$ , every (unbounded) interactive Turing machine  $\tilde{P}$ , and every string  $\tilde{\pi} \in \{0, 1\}^*$ ,

$$\Pr[(\tilde{P}(x), V^{\tilde{\pi}}(x)) = 1] \leq s(n)$$

(where  $n = |x|$ , and the probability is over the random coin tosses of  $V$ ).

- **Complexity:** The communication complexity of the protocol  $(P(x, w), V^\pi(x))$  is at most  $\ell(n)$ , and  $V$  reads at most  $q(n)$  bits of  $\pi$ .

**Notation.** We denote by  $\text{IPCP}(p, q, \ell, c, s)$  the set of all NP languages that have an interactive PCP with parameters  $(p, q, \ell, c, s)$ .

**Remark.** One can define interactive PCP where  $\pi$  is a string of symbols rather than bits (i.e., where the answer size is  $\geq 1$ ).

### 3 Ingredients

#### 3.1 Low Degree Extension

Fix a field  $\mathbb{F}$ , to be an extension field of  $\mathbb{GF}[2]$ . Fix a subset  $H \subset \mathbb{F}$  and an integer  $m$ . In what follows, we define the low degree extension of a  $k$ -element string  $(w_0, w_1, \dots, w_{k-1}) \in \mathbb{F}^k$  with respect to  $\mathbb{F}, H, m$ , where  $k = |H|^m$ .<sup>3</sup>

Fix  $\alpha : H^m \rightarrow \{0, 1, \dots, k-1\}$  to be any (efficiently computable) one-to-one function. In this paper, we take  $\alpha$  to be the lexicographic order of  $H^m$ . We can view  $(w_0, w_1, \dots, w_{k-1})$  as a function  $W : H^m \rightarrow \mathbb{F}$ , where  $W(z) = w_{\alpha(z)}$ . A basic fact is that there exists a unique extension of  $W$  into a function  $\tilde{W} : \mathbb{F}^m \rightarrow \mathbb{F}$ , such that  $\tilde{W}$  is an  $m$ -variate polynomial of degree at most  $|H| - 1$  in each variable. Moreover, the truth table of  $\tilde{W}$  can be computed from the truth table of  $W$  in time  $\text{poly}(|\mathbb{F}|^m)$ . The function  $\tilde{W}$  is called the *low degree extension* of  $w = (w_0, w_1, \dots, w_{k-1})$  with respect to  $\mathbb{F}, H, m$ , and is denoted by  $\text{LDE}_{\mathbb{F}, H, m}(w)$ .

**Proposition 3.1.** *There exists a Turing machine that takes as input an extension field  $\mathbb{F}$  of  $\mathbb{GF}[2]$ , a subset  $H \subset \mathbb{F}$ , an integer  $m$ , and a string  $w \in \mathbb{F}^k$ , where  $k = |H|^m$ , runs in time  $\leq \text{poly}(|\mathbb{F}|^m)$ , and outputs  $\text{LDE}_{\mathbb{F}, H, m}(w)$  (e.g., represented by its truth table).*

#### 3.2 Low Degree Test

Fix a finite field  $\mathbb{F}$ . Suppose that a verifier wishes to test whether a function  $\pi : \mathbb{F}^m \rightarrow \mathbb{F}$  is close to an  $m$ -variate polynomial of degree  $\leq d$  (think of  $d$  as significantly smaller than  $|\mathbb{F}|$ ). In this work, we think of a low degree test as an interactive proof for  $\pi$  being close to an  $m$ -variate polynomial of degree  $\leq d$ . This proof should be short (say, of size  $\leq \text{poly}(|\mathbb{F}|, m)$ ). The verifier has only oracle access to  $\pi$ , and is allowed to query  $\pi$  at only a few points (say, only one point).

In this work, we need a low degree test with sub-constant error. Three such tests exist in the literature: One due to Raz and Safra [RS97], one due to Arora and Sudan [AS97], and one due to Moshkovitz and Raz [MR05]. For the sake of convenience, we use the latter. The low degree test of [MR05] is described in Figure 1, and is denoted by  $(P_{\text{LDT}}(\pi), V_{\text{LDT}}^\pi)$ .

**Lemma 3.2.** *For any  $m \geq 3$  and  $1 \leq d \leq |\mathbb{F}|$ , the low degree test  $(P_{\text{LDT}}(\pi), V_{\text{LDT}}^\pi)$  described in Figure 1 has the following guarantees.*

- **Completeness:** *If  $\pi : \mathbb{F}^m \rightarrow \mathbb{F}$  is an  $m$ -variate polynomial of total degree  $\leq d$  then*

$$\Pr[(P_{\text{LDT}}(\pi), V_{\text{LDT}}^\pi) = 1] = 1$$

- **Soundness (decoding):** *For every  $\pi : \mathbb{F}^m \rightarrow \mathbb{F}$  and every (unbounded) interactive Turing machine  $\tilde{P}$ , if*

$$\Pr[(\tilde{P}(\pi), V_{\text{LDT}}^\pi) = 1] \geq \gamma$$

---

<sup>3</sup>We can actually work with  $k < |H|^m$  and pad the  $k$ -element string by zeros.

**Low Degree Test for  $\pi : \mathbb{F}^m \rightarrow \mathbb{F}$**

1. The verifier chooses uniformly and independently  $z_1, z_2, z_3 \in_R \mathbb{F}^m$ . If they are linearly dependent then he accepts. Otherwise, he sends the prover the triplet  $(z_1, z_2, z_3)$ .
2. The prover sends  $\eta : \mathbb{F}^3 \rightarrow \mathbb{F}$ , which is supposedly the function  $\pi$  restricted to the subspace  $U$  spanned by the vectors  $z_1, z_2, z_3$ . Namely,

$$\eta(\alpha_1, \alpha_2, \alpha_3) \stackrel{\text{def}}{=} \pi(\alpha_1 z_1 + \alpha_2 z_2 + \alpha_3 z_3).$$

3. The verifier checks that  $\eta$  is of degree at most  $d$ . If the check fails then the verifier rejects. Otherwise, the verifier chooses a random point  $z$  in the subspace  $U$ , by choosing uniformly  $\alpha_1, \alpha_2, \alpha_3 \in_R \mathbb{F}$  and setting  $z = \alpha_1 z_1 + \alpha_2 z_2 + \alpha_3 z_3$ . He queries the oracle  $\pi$  at the point  $z$ , and accepts if and only if

$$\eta(\alpha_1, \alpha_2, \alpha_3) = \pi(z).$$

Figure 1: Low degree test  $(P_{\text{LDT}}(\pi), V_{\text{LDT}}^\pi)$

then there exists an  $m$ -variate polynomial  $f : \mathbb{F}^m \rightarrow \mathbb{F}$  of total degree  $\leq d$ , such that  $\Pr_{z \in \mathbb{F}^m} [\pi(z) = f(z)] \geq \gamma - \epsilon$ , where

$$\epsilon \stackrel{\text{def}}{=} 2^{10} m^8 \sqrt{\frac{md}{|\mathbb{F}|}}.$$

- **Soundness (list-decoding):** For every  $\pi : \mathbb{F}^m \rightarrow \mathbb{F}$  and every  $\delta > \epsilon$  (where  $\epsilon$  is as above), there exist  $r \leq 2/\delta$  polynomials  $f_1, \dots, f_r : \mathbb{F}^m \rightarrow \mathbb{F}$  of total degree  $\leq d$ , such that: For every (unbounded) interactive Turing machine  $\tilde{P}$ ,

$$\Pr \left[ [(\tilde{P}(\pi), V_{\text{LDT}}^\pi) = 1] \wedge [\pi(z) \notin \{f_1(z), \dots, f_r(z)\}] \right] \leq \delta + \epsilon,$$

where  $z \in \mathbb{F}^m$  is the random element chosen in Step 3 of the low degree test.

- **Complexity:**  $P_{\text{LDT}}(\pi)$  is an interactive Turing machine, and  $V_{\text{LDT}}^\pi$  is a probabilistic interactive Turing machine with oracle access to  $\pi : \mathbb{F}^m \rightarrow \mathbb{F}$ . The prover  $P_{\text{LDT}}$  runs in time  $\leq \text{poly}(|\mathbb{F}|^m)$ . The verifier  $V_{\text{LDT}}^\pi$  runs in time  $\leq \text{poly}(|\mathbb{F}|, m)$  and queries the oracle  $\pi$  at a single point. The communication complexity is  $\leq \text{poly}(|\mathbb{F}|, m)$ .

We refer the reader to [MR05, MR06] for a proof of Lemma 3.2.

### 3.3 Point Test

Fix a finite field  $\mathbb{F}$ . Suppose that a verifier has oracle access to a function  $\pi : \mathbb{F}^m \rightarrow \mathbb{F}$  that is “close” to an  $m$ -variate polynomial  $f$  of total degree  $\leq d$  (think of  $d$  as significantly smaller than  $|\mathbb{F}|$ ). The verifier wishes to test whether  $f(0^m) = 1$ , and is required to be efficient, both in the running time and in the number of oracle queries. In Figure 2 we present a test that achieves this. We call this test the *point test*, and denote it by  $V_{\text{PT}}^\pi$ .

**Lemma 3.3.** For every  $d < |\mathbb{F}| - 1$  the point test described in Figure 2 has the following guarantees.

- **Completeness:** If  $\pi : \mathbb{F}^m \rightarrow \mathbb{F}$  is an  $m$ -variate polynomial of total degree  $\leq d$ , and  $\pi(0^m) = 1$  then

$$\Pr[V_{\text{PT}}^\pi = 1] = 1.$$

**Point Test for  $\pi : \mathbb{F}^m \rightarrow \mathbb{F}$**

1. The verifier chooses uniformly  $a \in_R \mathbb{F}^m \setminus \{0^m\}$ . Let  $\ell : \mathbb{F} \rightarrow \mathbb{F}^m$  be the line defined by  $\ell(t) \stackrel{\text{def}}{=} at$ .
2. The verifier chooses uniformly distinct  $t_1, \dots, t_{d+1} \in_R \mathbb{F} \setminus \{0\}$ , and sets  $z_i = at_i$  for every  $i \in [d+1]$ .
3. The verifier queries the oracle  $\pi$  at the points  $z_1, \dots, z_{d+1} \in \mathbb{F}^m$  and obtains values  $v_1, \dots, v_{d+1} \in \mathbb{F}$ .
4. The verifier computes by interpolation the univariate polynomial  $p : \mathbb{F} \rightarrow \mathbb{F}$  of degree  $\leq d$ , such that  $p(t_i) = v_i$  for every  $i \in [d+1]$ , and accepts if and only if  $p(0) = 1$ .

Figure 2: Point Test  $V_{\text{PT}}^\pi$

- **Soundness:** For every function  $\pi : \mathbb{F}^m \rightarrow \mathbb{F}$ , and every  $\gamma > 0$ , the following holds: If there exists an  $m$ -variate polynomial  $f : \mathbb{F}^m \rightarrow \mathbb{F}$  of total degree  $\leq d$  such that  $f(0^m) \neq 1$  and

$$\Pr_{z \in \mathbb{F}^m} [\pi(z) = f(z)] \geq 1 - \gamma,$$

then

$$\Pr[V_{\text{PT}}^\pi = 1] \leq (d+1) \left( \gamma + \frac{1}{|\mathbb{F}|^m} \right).$$

- **Complexity:** The verifier  $V_{\text{PT}}^\pi$  is a probabilistic Turing machine with oracle access to  $\pi : \mathbb{F}^m \rightarrow \mathbb{F}$ . It runs in time  $\leq \text{poly}(|\mathbb{F}|, m)$ , and queries the oracle  $\pi$  at  $d+1$  points.

**Proof of Lemma 3.3:** The completeness condition and the complexity condition follow immediately from the test description. As for the soundness, fix any function  $\pi : \mathbb{F}^m \rightarrow \mathbb{F}$  and any  $\gamma > 0$ . Assume that there exists an  $m$ -variate polynomial  $f : \mathbb{F}^m \rightarrow \mathbb{F}$  of total degree  $\leq d$  such that  $f(0^m) \neq 1$  and

$$\Pr_{z \in \mathbb{F}^m} [\pi(z) = f(z)] \geq 1 - \gamma.$$

This implies that

$$\Pr_{z \in \mathbb{F}^m \setminus \{0^m\}} [\pi(z) = f(z)] \geq 1 - \gamma - \frac{1}{|\mathbb{F}|^m}.$$

Thus,

$$\begin{aligned} \Pr[V_{\text{PT}}^\pi \neq 1] &\geq \\ \Pr[\forall i \in [d+1], \pi(z_i) \neq f(z_i)] &\geq \\ 1 - (d+1) \Pr_{z \in \mathbb{F}^m \setminus \{0^m\}} [\pi(z) \neq f(z)] &\geq \\ 1 - (d+1) \left( \gamma + \frac{1}{|\mathbb{F}|^m} \right). \end{aligned}$$

■

### 3.4 Interactive Sum-Check Protocol

Fix a finite field  $\mathbb{F}$ . In a sum-check protocol, a (not necessarily efficient) prover takes as input an  $m$ -variate polynomial  $f : \mathbb{F}^m \rightarrow \mathbb{F}$  of degree  $\leq d$  in each variable (think of  $d$  as significantly smaller than  $|\mathbb{F}|$ ). His goal is to convince a verifier that

$$\sum_{z \in H^m} f(z) = 0,$$

for a subset  $H \subset \mathbb{F}$ . The verifier only has oracle access to  $f$ , and is required to be efficient in both the running time and the number of oracle queries. In Figure 3, we review the standard sum-check protocol, as appeared for example in [LFKN90, S92]. We denote this protocol by  $(P_{\text{SC}}(f), V_{\text{SC}}^f)$ .

**Sum-Check Protocol for  $\sum_{t_1, \dots, t_m \in H} f(t_1, \dots, t_m) = 0$**

- In the first round,  $P$  computes the univariate polynomial  $g_1 : \mathbb{F} \rightarrow \mathbb{F}$  defined by

$$g_1(x) \stackrel{\text{def}}{=} \sum_{t_2, \dots, t_m \in H} f(x, t_2, \dots, t_m),$$

and sends  $g_1$  to  $V$ . Then,  $V$  checks that  $g_1 : \mathbb{F} \rightarrow \mathbb{F}$  is a univariate polynomial of degree at most  $d$ , and that

$$\sum_{x \in H} g_1(x) = 0.$$

If not  $V$  rejects. Otherwise,  $V$  chooses a random element  $c_1 \in_R \mathbb{F}$ , and sends  $c_1$  to  $P$ .

- In the  $i$ 'th round,  $P$  computes the univariate polynomial

$$g_i(x) \stackrel{\text{def}}{=} \sum_{t_{i+1}, \dots, t_m \in H} f(c_1, \dots, c_{i-1}, x, t_{i+1}, \dots, t_m),$$

and sends  $g_i$  to  $V$ . Then,  $V$  checks that  $g_i$  is a univariate polynomial of degree at most  $d$ , and that

$$\sum_{x \in H} g_i(x) = g_{i-1}(c_{i-1}).$$

If not  $V$  rejects. Otherwise,  $V$  chooses a random element  $c_i \in_R \mathbb{F}$ , and sends  $c_i$  to  $P$ .

- In the last round,  $P$  computes the univariate polynomial

$$g_m(x) \stackrel{\text{def}}{=} f(c_1, \dots, c_{m-1}, x),$$

and sends  $g_m$  to  $V$ . Finally,  $V$  checks that  $g_m$  is a univariate polynomial of degree at most  $d$ , and that

$$\sum_{x \in H} g_m(x) = g_{m-1}(c_{m-1}).$$

If not  $V$  rejects. Otherwise,  $V$  chooses a random element  $c_m \in_R \mathbb{F}$  and checks that

$$g_m(c_m) = f(c_1, \dots, c_m),$$

by querying the oracle at the point  $z = (c_1, \dots, c_m)$ .

Figure 3: Sum-check protocol  $(P_{\text{SC}}(f), V_{\text{SC}}^f)$  [LFKN90, S92]

**Lemma 3.4.** *Let  $f : \mathbb{F}^m \rightarrow \mathbb{F}$  be an  $m$ -variate polynomial of degree at most  $d$  in each variable, where  $d < |\mathbb{F}|$ . The sum-check protocol  $(P_{\text{SC}}(f), V_{\text{SC}}^f)$ , described in Figure 3, satisfies the following properties.*

- **Completeness:** *If  $\sum_{z \in H^m} f(z) = 0$  then*

$$\Pr \left[ \left( P_{\text{SC}}(f), V_{\text{SC}}^f \right) = 1 \right] = 1.$$

- **Soundness:** *If  $\sum_{z \in H^m} f(z) \neq 0$  then for every (unbounded) interactive Turing machine  $\tilde{P}$ ,*

$$\Pr \left[ \left( \tilde{P}(f), V_{\text{SC}}^f \right) = 1 \right] \leq \frac{md}{|\mathbb{F}|}.$$

- **Complexity:**  *$P_{\text{SC}}(f)$  is an interactive Turing machine, and  $V_{\text{SC}}^f$  is a probabilistic interactive Turing machine with oracle access to  $f : \mathbb{F}^m \rightarrow \mathbb{F}$ . The prover  $P_{\text{SC}}(f)$  runs in time  $\leq \text{poly}(|\mathbb{F}|^m)$ . The verifier  $V_{\text{SC}}^f$  runs in time  $\leq \text{poly}(|\mathbb{F}|, m)$ , and queries the oracle  $f$  at a single point. The communication complexity is  $\leq \text{poly}(|\mathbb{F}|, m)$ .*

**Proof of Lemma 3.4:** The completeness condition and the complexity condition follow immediately from the protocol description. As for the soundness, let  $f : \mathbb{F}^m \rightarrow \mathbb{F}$  be a polynomial of degree at most  $d$  in each variable, such that  $\sum_{z \in H^m} f(z) \neq 0$ . Assume for the sake of contradiction that there exists a cheating prover  $\tilde{P}$  for which

$$s \stackrel{\text{def}}{=} \Pr \left[ \left( \tilde{P}(f), V_{\text{SC}}^f \right) = 1 \right] > \frac{md}{|\mathbb{F}|}.$$

Recall that in the sum-check protocol the prover sends  $m$  univariate polynomials  $g_1, \dots, g_m$ , and the verifier sends  $m - 1$  random field elements  $c_1, \dots, c_{m-1} \in \mathbb{F}$ . For every  $i \in [m]$ , let  $A_i$  denote the event that

$$g_i(x) = \sum_{t_{i+1}, \dots, t_m \in H} f(c_1, \dots, c_{i-1}, x, t_{i+1}, \dots, t_m).$$

Let  $S$  denote the event that  $\left( \tilde{P}(f), V_{\text{SC}}^f \right) = 1$ . Notice that  $\Pr[S | A_1 \wedge \dots \wedge A_m] = 0$ . We will reach a contradiction by proving that

$$\Pr[S | A_1 \wedge \dots \wedge A_m] \geq s - \frac{md}{|\mathbb{F}|}.$$

To this end, we prove by (reverse) induction that for every  $j \in [m]$ ,

$$\Pr[S | A_j \wedge \dots \wedge A_m] \geq s - \frac{(m - j + 1)d}{|\mathbb{F}|}. \quad (1)$$

For  $j = m$ ,

$$s = \Pr[S] \leq \Pr[S | \neg(A_m)] + \Pr[S | A_m] \leq \frac{d}{|\mathbb{F}|} + \Pr[S | A_m],$$

where the latter inequality follows from the fact that every two distinct univariate polynomials of degree  $\leq d$  over  $\mathbb{F}$  agree in at most  $\frac{d}{|\mathbb{F}|}$  points. Thus,

$$\Pr[S|A_m] \geq s - \frac{d}{|\mathbb{F}|}.$$

Assume that Equation (1) holds for  $j$ , and we will show that it holds for  $j - 1$ .

$$\begin{aligned} s - \frac{(m - j + 1)d}{|\mathbb{F}|} &\leq \Pr[S|A_j \wedge \dots \wedge A_m] \leq \\ &\Pr[S|\neg(A_{j-1}) \wedge A_j \wedge \dots \wedge A_m] + \Pr[S|A_{j-1} \wedge A_j \wedge \dots \wedge A_m] \leq \\ &\frac{d}{|\mathbb{F}|} + \Pr[S|A_{j-1} \wedge \dots \wedge A_m], \end{aligned}$$

which implies that

$$\Pr[S|A_{j-1} \wedge \dots \wedge A_m] \geq s - \frac{(m - (j - 1) + 1)d}{|\mathbb{F}|},$$

as desired.  $\blacksquare$

We note that our interactive PCP protocol makes use of this sum-check protocol with  $\mathbb{F}^m$  which is of size super polynomial. Thus, at first it seems like the prover in our protocol must run in super-polynomial time. However, we bypass this barrier by exploiting special properties of the function  $f$ , on which we apply the sum-check protocol, and obtain a prover with polynomial running time. See Lemma 4.5 in Section 4 for further details.

## 4 Interactive PCP for Arithmetic Formulas: Part I

Fix a parameter  $d \in \mathbb{N}$ . Fix a field  $\mathbb{F}$  to be an extension field of  $\mathbb{GF}[2]$ , such that  $|\mathbb{F}| > d$ . Fix a subset  $H \subset \mathbb{F}$  and an integer  $m$ . Let  $(w_1, \dots, w_k) \in \mathbb{F}^k$  be a string of length  $k$ , where  $k + 1 = |H|^m$ .<sup>4</sup> Let  $\tilde{W} : \mathbb{F}^m \rightarrow \mathbb{F}$  be the low degree extension of  $(1, w_1, \dots, w_k)$  with respect to  $\mathbb{F}, H, m$  (as defined in Subsection 3.1). Let  $\Psi : \mathbb{F}^k \rightarrow \mathbb{F}$  be an arithmetic formula of syntactic degree  $\leq d$  (for the definition of syntactic degree see Definition 2 below). Assume w.l.o.g. that  $|\Psi| \geq k, d$ . Assume that addition and multiplication over  $\mathbb{F}$  are operations of complexity  $\leq \text{poly}(\log |\mathbb{F}|)$ .

We show how a prover can interactively prove to a verifier that  $\Psi(w) = 0$ . The formula  $\Psi$  is known to both the prover and the verifier. The string  $w \in \mathbb{F}^k$  is known only to the prover, and the verifier is given oracle access to the low degree extension  $\tilde{W}$ . The communication complexity is  $\leq \text{poly}(|\mathbb{F}|, m)$ , and the verifier queries the oracle  $\tilde{W}$  at exactly  $d$  points. Moreover, the prover and the verifier are both probabilistic interactive Turing machines that run in time  $\leq \text{poly}(|\Psi|, |\mathbb{F}|^m)$ . Note that if  $\mathbb{F}$  satisfies  $|\mathbb{F}| \leq \text{poly}(|H|)$ , then their running time is actually  $\leq \text{poly}(|\Psi|)$ .

Throughout this section, it will be convenient to assume that all the arithmetic formulas are *syntactically homogeneous*.

### 4.1 Syntactically Homogeneous Arithmetic Formulas

We start by defining the *syntactic degree* of an arithmetic formula, a notion that is used extensively in this section and throughout the paper. Intuitively, the syntactic degree of an arithmetic formula is its degree (as a polynomial) if one ignores cancelations of monomials.

**Definition 2.** Let  $\Psi : \mathbb{F}^k \rightarrow \mathbb{F}$  be an arithmetic formula. Its **syntactic degree** is defined by recursion on  $\Psi$ . If  $\Psi$  is a constant, then its syntactic degree is 0. If  $\Psi$  is a variable, then its

<sup>4</sup>As mentioned in Subsection 3.1, we can also work with  $k + 1 < |H|^m$  and pad it by zeros.



syntactic degree is 1. If  $\Psi$  is of the form  $\Psi = \Psi_1 + \Psi_2$ , then its syntactic degree is the maximum of the syntactic degrees of  $\Psi_1$  and  $\Psi_2$ . If  $\Psi$  is of the form  $\Psi = \Psi_1 \cdot \Psi_2$ , then its syntactic degree is the sum of the syntactic degrees of  $\Psi_1$  and  $\Psi_2$ .

**Definition 3.** An arithmetic formula  $\Phi : \mathbb{F}^k \rightarrow \mathbb{F}$  is **syntactically homogeneous** if for all its sub-formulas of the form  $\Phi_1 + \Phi_2$ , the syntactic degrees of  $\Phi_1$  and  $\Phi_2$  are the same.

For convenience, from now on we abuse notation: whenever we refer to a homogeneous formula, we actually mean a syntactically homogeneous formula.

**Converting arithmetic formulas to homogeneous ones.** For a given degree  $d$  and an arithmetic formula  $\Psi : \mathbb{F}^k \rightarrow \mathbb{F}$  of syntactic degree  $d_\Psi \leq d$ , we show how to convert  $\Psi$  into a homogeneous arithmetic formula of syntactic degree exactly  $d$ . Loosely speaking, this is done by adding a dummy variable  $x_0$ . We think of  $x_0$  as the constant 1. For each sub-formula of the form  $\Psi_1 + \Psi_2$ , if the syntactic degree of  $\Psi_1$  is  $d_1$ , and the syntactic degree of  $\Psi_2$  is  $d_2 > d_1$ , then we multiply  $\Psi_1$  by  $x_0^{d_2-d_1}$ . This increases the syntactic degree of  $\Psi_1$  to be  $d_2$ . The case that  $d_1 > d_2$  is treated in a similar manner. This results with a homogeneous formula of syntactic degree  $d_\Psi \leq d$ . We then multiply this formula by  $x_0^{d-d_\Psi}$ , and get a homogeneous formula of syntactic degree exactly  $d$ .

This conversion is formalized below.

**Proposition 4.1.** *There exists a Turing machine  $\mathcal{M}$  that takes as input an integer  $d \in \mathbb{N}$  and an arithmetic formula  $\Psi : \mathbb{F}^k \rightarrow \mathbb{F}$  of syntactic degree  $d_\Psi \leq d$ . The Turing machine  $\mathcal{M}(d, \Psi)$  runs in time  $\leq \text{poly}(d, |\Psi|)$ . It outputs a homogeneous arithmetic formula  $\Phi : \mathbb{F}^{k+1} \rightarrow \mathbb{F}$  of syntactic degree  $d$ , such that for every  $x_1, \dots, x_k \in \mathbb{F}$ ,*

$$\Phi(1, x_1, \dots, x_k) = \Psi(x_1, \dots, x_k).$$

**Proof of Proposition 4.1:** Fix an integer  $d \in \mathbb{N}$  and an arithmetic formula  $\Psi : \mathbb{F}^k \rightarrow \mathbb{F}$  of syntactic degree  $d_\Psi \leq d$ . We begin by defining  $\mathcal{M}$  for the case that  $d_\Psi = d$ .

The Turing machine  $\mathcal{M}(d_\Psi, \Psi)$  works recursively on  $\Psi$ . If  $d_\Psi = 1$ , i.e.,  $\Psi(x_1, \dots, x_k) = c_0 + \sum_{i=1}^k c_i x_i$ , then  $\mathcal{M}(d_\Psi, \Psi)$  outputs

$$\Phi(x_0, x_1, \dots, x_k) \stackrel{\text{def}}{=} c_0 x_0 + \sum_{i=1}^k c_i x_i = \sum_{i=0}^k c_i x_i.$$

If  $d_\Psi > 1$ , then we distinguish between two cases.

1. **Case 1:**  $\Psi = \Psi_1 \cdot \Psi_2$ . Denote by  $d_1$  and  $d_2$  the syntactic degrees of  $\Psi_1$  and  $\Psi_2$ , respectively. The Turing machine  $\mathcal{M}(d_\Psi, \Psi)$  runs  $\mathcal{M}(d_1, \Psi_1)$  to compute  $\Phi_1 : \mathbb{F}^{k+1} \rightarrow \mathbb{F}$ , which is the homogeneous formula of syntactic degree  $d_1$  corresponding to  $\Psi_1$ . Similarly, it runs  $\mathcal{M}(d_2, \Psi_2)$  to compute  $\Phi_2 : \mathbb{F}^{k+1} \rightarrow \mathbb{F}$ , which is the homogeneous formula of syntactic degree  $d_2$  corresponding to  $\Psi_2$ . It outputs  $\Phi \stackrel{\text{def}}{=} \Phi_1 \cdot \Phi_2$ . Note that since  $\Phi_1$  and  $\Phi_2$  are homogenous then so is  $\Phi$ . Its degree is  $d_1 + d_2 = d_\Psi$ . Moreover, for every  $x_1, \dots, x_k \in \mathbb{F}$ ,

$$\begin{aligned} \Phi(1, x_1, \dots, x_k) &= \Phi_1(1, x_1, \dots, x_k) \cdot \Phi_2(1, x_1, \dots, x_k) = \\ &= \Psi_1(x_1, \dots, x_k) \cdot \Psi_2(x_1, \dots, x_k) = \\ &= \Psi(x_1, \dots, x_k). \end{aligned}$$

2. **Case 2:**  $\Psi = \Psi_1 + \Psi_2$ . Denote by  $d_1$  and  $d_2$  the syntactic degrees of  $\Psi_1$  and  $\Psi_2$ , respectively. The Turing machine  $\mathcal{M}(d_\Psi, \Psi)$  runs  $\mathcal{M}(d_1, \Psi_1)$  to compute  $\Phi_1 : \mathbb{F}^{k+1} \rightarrow \mathbb{F}$ , which is the homogeneous formula of syntactic degree  $d_1$  corresponding to  $\Psi_1$ . Similarly, it runs  $\mathcal{M}(d_2, \Psi_2)$  to compute  $\Phi_2 : \mathbb{F}^{k+1} \rightarrow \mathbb{F}$ , which is the homogeneous formula of syntactic degree  $d_2$  corresponding to  $\Psi_2$ . If  $d_1 = d_2$ , then it outputs  $\Phi \stackrel{\text{def}}{=} \Phi_1 + \Phi_2$ . If  $d_1 > d_2$ , then it outputs  $\Phi \stackrel{\text{def}}{=} \Phi_1 + x_0^{d_1-d_2}\Phi_2$ . If  $d_2 > d_1$ , then it outputs  $\Phi \stackrel{\text{def}}{=} x_0^{d_2-d_1}\Phi_1 + \Phi_2$ . Note that  $\Phi$  is homogeneous of syntactic degree  $\max\{d_1, d_2\} = d_\Psi$ . As before, for every  $x_1, \dots, x_k \in \mathbb{F}$ ,

$$\Phi(1, x_1, \dots, x_k) = \Psi(x_1, \dots, x_k).$$

We next consider the case that  $d_\Psi < d$ . In this case  $\mathcal{M}(d, \Psi)$  runs  $\mathcal{M}(d_\Psi, \Psi)$  to compute  $\Phi_0$ , and outputs  $\Phi \stackrel{\text{def}}{=} x_0^{d-d_\Psi} \cdot \Phi_0$ . This increases its syntactic degree from  $d_\Psi$  to  $d$ . For every  $x_1, \dots, x_k \in \mathbb{F}$ ,

$$\Phi(1, x_1, \dots, x_k) = 1 \cdot \Phi_0(1, x_1, \dots, x_k) = \Psi(x_1, \dots, x_k).$$

■

## 4.2 Definition of $\hat{\Phi}$ and its Properties

As in the beginning of Section 4, we fix  $\mathbb{F}$  to be an extension field of  $\mathbb{GF}[2]$ . We fix a subset  $H \subset \mathbb{F}$  and an integer  $m$ . We let  $k+1 = |H|^m$ . In this subsection, for every homogeneous arithmetic formula  $\Phi : \mathbb{F}^{k+1} \rightarrow \mathbb{F}$  of syntactic degree  $d$ , we define a corresponding function  $\hat{\Phi} : \mathbb{F}^{md} \rightarrow \mathbb{F}$ .

This function is defined in a recursive manner. We start with the base case. If  $\Phi : \mathbb{F}^{k+1} \rightarrow \mathbb{F}$  is a homogeneous arithmetic formula of syntactic degree  $d = 1$ , i.e.,  $\Phi(x_0, x_1, \dots, x_k) = \sum_{i=0}^k c_i x_i$ , then we define  $\hat{\Phi} : \mathbb{F}^m \rightarrow \mathbb{F}$  to be the low degree extension of  $(c_0, c_1, \dots, c_k)$  with respect to  $\mathbb{F}, H, m$ . Namely,  $\hat{\Phi} = \text{LDE}_{\mathbb{F}, H, m}(c_0, c_1, \dots, c_k)$ . (We refer the reader to Subsection 3.1 for the definition of low degree extension). If  $\Phi : \mathbb{F}^{k+1} \rightarrow \mathbb{F}$  is a homogeneous arithmetic formula of syntactic degree  $d > 1$ , we define  $\hat{\Phi} : \mathbb{F}^{md} \rightarrow \mathbb{F}$  recursively, as follows:

1. **Case 1:** The top gate is an addition gate, i.e.,  $\Phi = \Phi_1 + \Phi_2$ . The fact that  $\Phi$  is homogeneous implies that the syntactic degree of both  $\Phi_1$  and  $\Phi_2$  is  $d$ , and thus  $\hat{\Phi}_1, \hat{\Phi}_2 : \mathbb{F}^{md} \rightarrow \mathbb{F}$ . For every  $z \in \mathbb{F}^{md}$ ,

$$\hat{\Phi}(z) \stackrel{\text{def}}{=} \hat{\Phi}_1(z) + \hat{\Phi}_2(z).$$

2. **Case 2:** The top gate is a multiplication gate, i.e.,  $\Phi = \Phi_1 \cdot \Phi_2$ . Denote by  $d_1$  and  $d_2$  the syntactic degrees of  $\Phi_1$  and  $\Phi_2$ , respectively. Thus,  $\hat{\Phi}_1 : \mathbb{F}^{md_1} \rightarrow \mathbb{F}$  and  $\hat{\Phi}_2 : \mathbb{F}^{md_2} \rightarrow \mathbb{F}$ . Note that  $d_1 + d_2 = d$ . For every  $z \in \mathbb{F}^{md}$ ,

$$\hat{\Phi}(z) \stackrel{\text{def}}{=} \hat{\Phi}_1(z_1) \cdot \hat{\Phi}_2(z_2),$$

where  $z = (z_1, z_2) \in \mathbb{F}^{md_1} \times \mathbb{F}^{md_2}$ .

**Claim 4.2.** *If  $\Phi : \mathbb{F}^{k+1} \rightarrow \mathbb{F}$  is a homogenous arithmetic formula of syntactic degree  $d$ , then  $\hat{\Phi} : \mathbb{F}^{md} \rightarrow \mathbb{F}$  is a polynomial of degree  $< m|H|d$ .*

**Proof of Claim 4.2:** The proof is by induction on  $d$ . If  $d = 1$ , then the claim follows from Subsection 3.1. Assume that the claim holds for every degree  $< d$  and we prove that it also holds for degree  $d$ . Let  $\Phi = \sum_{i=1}^l \Phi_{i,1} \cdot \Phi_{i,2}$ , where the syntactic degrees of  $\Phi_{i,1}$  and  $\Phi_{i,2}$ , denoted by  $d_{i,1}$  and  $d_{i,2}$ , respectively, are both strictly smaller than  $d$ . From the induction hypothesis,  $\hat{\Phi}_{i,1} : \mathbb{F}^{md_{i,1}} \rightarrow \mathbb{F}$  is of degree  $< m|H|d_{i,1}$ , and  $\hat{\Phi}_{i,2} : \mathbb{F}^{md_{i,2}} \rightarrow \mathbb{F}$  is of degree  $< m|H|d_{i,2}$ . This implies that the degree of  $\hat{\Phi}$  is  $< m|H|d$ . ■

Let  $\alpha : H^m \rightarrow \{0, 1, \dots, k\}$  be the lexicographic order of  $H^m$ .

**Claim 4.3.** Let  $\Phi : \mathbb{F}^{k+1} \rightarrow \mathbb{F}$  be a homogeneous arithmetic formula of syntactic degree  $d$ . Let  $w$  be any string in  $\mathbb{F}^{k+1}$ , and let  $\tilde{W} : \mathbb{F}^m \rightarrow \mathbb{F}$  be the low degree extension of  $w$  with respect to  $\mathbb{F}, H, m$ . Then,

$$\sum_{z^1, \dots, z^d \in H^m} \tilde{W}(z^1) \cdot \dots \cdot \tilde{W}(z^d) \cdot \hat{\Phi}(z^1, \dots, z^d) = \Phi(w). \quad (2)$$

**Proof of Claim 4.3:** The proof is by recursion on  $\Phi$ . We start with the base case. If  $\Phi : \mathbb{F}^{k+1} \rightarrow \mathbb{F}$  is a homogeneous arithmetic formula of syntactic degree  $d = 1$ , i.e.,  $\Phi(w) = \sum_{i=0}^k c_i w_i$ , then

$$\sum_{z \in H^m} \tilde{W}(z) \cdot \hat{\Phi}(z) = \sum_{z \in H^m} w_{\alpha(z)} \cdot c_{\alpha(z)} = \sum_{i=0}^k c_i w_i = \Phi(w).$$

If  $\Phi : \mathbb{F}^{k+1} \rightarrow \mathbb{F}$  is a homogeneous arithmetic formula of syntactic degree  $d > 1$  then we distinguish between two cases.

- **Case 1:**  $\Phi = \Phi_1 + \Phi_2$ . Recall that since  $\Phi$  is homogeneous, the syntactic degree of both  $\Phi_1$  and  $\Phi_2$  is exactly  $d$ , and thus  $\hat{\Phi}_1, \hat{\Phi}_2 : \mathbb{F}^{md} \rightarrow \mathbb{F}$ . Assume that Equation (2) holds for  $\Phi_1$  and  $\Phi_2$ . Then,

$$\begin{aligned} & \sum_{z^1, \dots, z^d \in H^m} \tilde{W}(z^1) \cdot \dots \cdot \tilde{W}(z^d) \cdot \hat{\Phi}(z^1, \dots, z^d) = \\ & \sum_{z^1, \dots, z^d \in H^m} \tilde{W}(z^1) \cdot \dots \cdot \tilde{W}(z^d) \cdot \hat{\Phi}_1(z^1, \dots, z^d) + \\ & \sum_{z^1, \dots, z^d \in H^m} \tilde{W}(z^1) \cdot \dots \cdot \tilde{W}(z^d) \cdot \hat{\Phi}_2(z^1, \dots, z^d) = \\ & \Phi_1(w) + \Phi_2(w) = \Phi(w). \end{aligned}$$

- **Case 2:** If  $\Phi = \Phi_1 \cdot \Phi_2$ . We denote the syntactic degrees of  $\Phi_1$  and  $\Phi_2$  by  $d_1$  and  $d_2$ , respectively (where  $d_1 + d_2 = d$ ). Assume that Equation (2) holds for  $\Phi_1$  and  $\Phi_2$ . Then,

$$\begin{aligned} & \sum_{z^1, \dots, z^d \in H^m} \tilde{W}(z^1) \cdot \dots \cdot \tilde{W}(z^d) \cdot \hat{\Phi}(z^1, \dots, z^d) = \\ & \left( \sum_{z^1, \dots, z^{d_1} \in H^m} \tilde{W}(z^1) \cdot \dots \cdot \tilde{W}(z^{d_1}) \cdot \hat{\Phi}_1(z^1, \dots, z^{d_1}) \right) \cdot \\ & \left( \sum_{z^1, \dots, z^{d_2} \in H^m} \tilde{W}(z^1) \cdot \dots \cdot \tilde{W}(z^{d_2}) \cdot \hat{\Phi}_2(z^1, \dots, z^{d_2}) \right) = \\ & \Phi_1(w) \cdot \Phi_2(w) = \Phi(w). \end{aligned}$$

■

**Claim 4.4.** *There exists a Turing machine that takes as input a homogeneous arithmetic formula  $\Phi : \mathbb{F}^{k+1} \rightarrow \mathbb{F}$  of syntactic degree  $d$  and a tuple  $(z^1, \dots, z^d) \in (\mathbb{F}^m)^d$ , runs in time  $\leq \text{poly}(|\Phi|, |\mathbb{F}|^m)$ , and outputs  $\hat{\Phi}(z^1, \dots, z^d)$ .*

**Proof of Claim 4.4:** Let  $\Phi : \mathbb{F}^{k+1} \rightarrow \mathbb{F}$  be a homogeneous arithmetic formula of syntactic degree  $d$ , and let  $z \in \mathbb{F}^{md}$ . We construct a Turing machine  $\mathcal{M}$  that on input  $(\Phi, z)$  outputs  $\hat{\Phi}(z)$ . The Turing machine  $\mathcal{M}(\Phi, z)$  works recursively on  $\Phi$ . If  $d = 1$  then  $\mathcal{M}(\Phi, z)$  computes  $\hat{\Phi}(z)$  by computing the low degree extension of its  $k+1$  coefficients on the point  $z$ . From Proposition 3.1, this can be done in time  $\leq \text{poly}(|\mathbb{F}|^m)$ . For  $d > 1$ ,  $\mathcal{M}(\Phi, z)$  computes  $\hat{\Phi}(z)$  as follows:

1. If  $\Phi = \Phi_1 + \Phi_2$  then  $\mathcal{M}(\Phi, z)$  computes  $\hat{\Phi}_1(z)$  and  $\hat{\Phi}_2(z)$ , and adds the results.
2. If  $\Phi = \Phi_1 \cdot \Phi_2$  then let  $d_1$  and  $d_2$  be the syntactic degrees of  $\Phi_1$  and  $\Phi_2$ , respectively.  $\mathcal{M}(\Phi, z)$  computes  $\hat{\Phi}(z)$ , where  $z = (z_1, z_2) \in \mathbb{F}^{md_1} \times \mathbb{F}^{md_2}$ , by computing  $\hat{\Phi}_1(z_1)$  and  $\hat{\Phi}_2(z_2)$ , and multiplying the results.

### 4.3 The Protocol

**Parameters.**  $d, \mathbb{F}, H, m$ , as defined in the beginning of Section 4.

**Input.** Both the prover and the verifier take as input an arithmetic formula  $\Psi : \mathbb{F}^k \rightarrow \mathbb{F}$  of syntactic degree  $\leq d$ , where  $k+1 = |H|^m$ . The prover takes an additional input  $w = (w_1, \dots, w_k) \in \mathbb{F}^k$ . Fix  $w_0 \stackrel{\text{def}}{=} 1$ . The verifier is given oracle access to  $\pi \stackrel{\text{def}}{=} \text{LDE}_{\mathbb{F}, H, m}(w_0, w_1, \dots, w_k)$ , which is *guaranteed* to be the low degree extension of  $(w_0, w_1, \dots, w_k)$  with respect to  $\mathbb{F}, H, m$ .

We show how the prover can interactively prove to the verifier that  $\Psi(w) = 0$ . The prover and verifier run in time  $\leq \text{poly}(|\Psi|, |\mathbb{F}|^m)$ . The verifier queries the oracle  $\pi$  at  $d$  points. The communication complexity is  $\leq \text{poly}(|\mathbb{F}|, m)$ .

The protocol can be divided into two parts.

#### 1. Converting $\Psi$ to a homogeneous formula.

Both the prover and the verifier (separately) convert  $\Psi$  to a homogeneous formula  $\Phi : \mathbb{F}^{k+1} \rightarrow \mathbb{F}$  of syntactic degree  $d$ , as described in Subsection 4.1. From Proposition 4.1,

$$\Phi(w_0, w_1, \dots, w_k) = \Psi(w_1, \dots, w_k).$$

#### 2. Proving that $\Phi(w_0, w_1, \dots, w_k) = 0$ .

Let  $f_{\Phi, \pi} : \mathbb{F}^{md} \rightarrow \mathbb{F}$  be the function defined as follows: For every  $z^1, \dots, z^d \in \mathbb{F}^m$ ,

$$f_{\Phi, \pi}(z^1, \dots, z^d) \stackrel{\text{def}}{=} \pi(z^1) \cdot \dots \cdot \pi(z^d) \cdot \hat{\Phi}(z^1, \dots, z^d).$$

Claim 4.3 implies that,

$$\sum_{z^1, \dots, z^d \in H^m} f_{\Phi, \pi}(z^1, \dots, z^d) = \Phi(w_0, w_1, \dots, w_k).$$

In order to prove that  $\Phi(w_0, w_1, \dots, w_k) = 0$ , the prover and verifier engage in the interactive sum-check protocol  $(P_{\text{SC}}(f_{\Phi, \pi}), V_{\text{SC}}^{(f_{\Phi, \pi})})$ , described in Subsection 3.4, for proving that

$$\sum_{t_1, \dots, t_{md} \in H} f_{\Phi, \pi}(t_1, \dots, t_{md}) = 0.$$

If  $(P_{\text{SC}}(f_{\Phi, \pi}), V_{\text{SC}}^{(f_{\Phi, \pi})}) = 1$  then the verifier accepts. Otherwise, the verifier rejects. Note that the verifier can compute  $f_{\Phi, \pi}$  on a given input by querying the oracle  $\pi$  at  $d$  points. Thus,  $\pi$  can serve as an oracle to  $f_{\Phi, \pi}$ .

We will analyze the properties of this protocol in Theorem 1 below. Note that applying the sum-check protocol naively results with the prover running in time  $|\mathbb{F}|^{md} > k^d$  (i.e., time exponential in  $d$ ). To reduce the running time of the prover, we will use the Turing machine  $\mathcal{M}$  defined in the following lemma, when running the sum-check protocol.

**Lemma 4.5.** *There exists a Turing machine  $\mathcal{M}$  that takes as input a homogeneous arithmetic formula  $\Phi : \mathbb{F}^{k+1} \rightarrow \mathbb{F}$  of syntactic degree  $d$ , a function  $\pi : \mathbb{F}^m \rightarrow \mathbb{F}$  (e.g., by its truth table), an integer  $i \in \{0, 1, \dots, md\}$ , and elements  $c_1, \dots, c_i \in \mathbb{F}$ , runs in time  $\text{poly}(|\Phi|, |\mathbb{F}|^m)$ , and outputs*

$$\sum_{t_{i+1}, \dots, t_{md} \in H} f_{\Phi, \pi}(c_1, \dots, c_i, t_{i+1}, \dots, t_{md}). \quad (3)$$

Note that in the sum-check protocol, the prover only needs to compute expressions of the same form as Expression (3). The prover can hence use the Turing machine  $\mathcal{M}$  from Lemma 4.5. We denote the protocol described above by

$$(P_1(w, \Psi), V_1^\pi(\Psi)).$$

A succinct description of it appears in Figure 4.

**Proof of Lemma 4.5:** By definition,

$$f_{\Phi, \pi}(c_1, \dots, c_i, t_{i+1}, \dots, t_{md}) = \pi^d(c_1, \dots, c_i, t_{i+1}, \dots, t_{md}) \cdot \hat{\Phi}(c_1, \dots, c_i, t_{i+1}, \dots, t_{md}),$$

where,

$$\pi^d(x_1, \dots, x_{md}) \stackrel{\text{def}}{=} \pi(x_1, \dots, x_m) \cdot \pi(x_{m+1}, \dots, x_{2m}) \cdot \dots \cdot \pi(x_{m(d-1)+1}, \dots, x_{md}).$$

We construct a Turing machine  $\mathcal{M}$  for the task described in Lemma 4.5. The Turing machine  $\mathcal{M}(\Phi, \pi, i, c_1, \dots, c_i)$  works recursively on  $\Phi$ . We start with the base case. If  $\Phi$  is of syntactic degree  $d = 1$ , then Claim 4.4 implies that  $\mathcal{M}(\Phi, \pi, i, c_1, \dots, c_i)$  can compute Expression (3) in time  $\leq \text{poly}(|\Phi|, |\mathbb{F}|^m)$ . For a general syntactic degree  $d > 1$ , the Turing machine  $\mathcal{M}(\Phi, \pi, i, c_1, \dots, c_i)$  computes Expression (3) recursively, as follows.

- **Case 1:**  $\Phi = \Phi_1 + \Phi_2$ . The fact that  $\Phi$  is homogeneous implies that the syntactic degree of both  $\Phi_1$  and  $\Phi_2$  is exactly  $d$ . Thus,  $\hat{\Phi}_1, \hat{\Phi}_2 : \mathbb{F}^{md} \rightarrow \mathbb{F}$ .

$$\begin{aligned} & \sum_{t_{i+1}, \dots, t_{md} \in H} \pi^d(c_1, \dots, c_i, t_{i+1}, \dots, t_{md}) \cdot \hat{\Phi}(c_1, \dots, c_i, t_{i+1}, \dots, t_{md}) = \\ & \sum_{t_{i+1}, \dots, t_{md} \in H} \pi^d(c_1, \dots, c_i, t_{i+1}, \dots, t_{md}) \cdot \hat{\Phi}_1(c_1, \dots, c_i, t_{i+1}, \dots, t_{md}) + \\ & \sum_{t_{i+1}, \dots, t_{md} \in H} \pi^d(c_1, \dots, c_i, t_{i+1}, \dots, t_{md}) \cdot \hat{\Phi}_2(c_1, \dots, c_i, t_{i+1}, \dots, t_{md}) \end{aligned}$$

**Interactive Proof for  $\Psi(w) = 0$**

**Parameters:**  $d, \mathbb{F}, H, m$  (as described in the beginning of Section 4).

**Common Input to  $P$  and  $V$ :** arithmetic formula  $\Psi : \mathbb{F}^k \rightarrow \mathbb{F}$  of syntactic degree  $\leq d$ , where  $k + 1 = |H|^m$ .

**Private Input to  $P$ :**  $(w_1, \dots, w_k) \in \mathbb{F}^k$

**Oracle to  $V$ :**  $\pi \stackrel{\text{def}}{=} \text{LDE}_{\mathbb{F}, H, m}(1, w_1, \dots, w_k)$

$P, V$ :

Compute

$$\Phi : \mathbb{F}^{k+1} \rightarrow \mathbb{F},$$

which is the homogeneous arithmetic formula corresponding to  $d$  and  $\Psi$ , as described in Subsection 4.1.

$P, V$

Let  $f_{\Phi, \pi} : \mathbb{F}^{md} \rightarrow \mathbb{F}$ , defined as follows: for every  $z^1, \dots, z^d \in \mathbb{F}^m$ ,

$$f_{\Phi, \pi}(z^1, \dots, z^d) \stackrel{\text{def}}{=} \pi(z^1) \cdot \dots \cdot \pi(z^d) \cdot \hat{\Phi}(z^1, \dots, z^d).$$

$P \rightleftharpoons V$ :

Run the sum-check protocol  $(P_{\text{SC}}(f_{\Phi, \pi}), V_{\text{SC}}^{(f_{\Phi, \pi})})$  described in Figure 3, for proving that

$$\sum_{t_1, \dots, t_{md} \in H} f_{\Phi, \pi}(t_1, \dots, t_{md}) = 0.$$

$P$  uses the Turing machine from Lemma 4.5 to run this sum-check efficiently.  $V$  uses the oracle  $\pi$  to simulate the oracle  $f_{\Phi, \pi}$ .

Figure 4:  $(P_1(w, \Psi), V_1^\pi(\Psi))$

In this case,  $\mathcal{M}(\Phi, \pi, i, c_1, \dots, c_i)$  computes Expression (3) recursively: It computes separately the part that corresponds to  $\hat{\Phi}_1$  and the part that corresponds to  $\hat{\Phi}_2$ , and then adds the results.

• **Case 2:**  $\Phi = \Phi_1 \cdot \Phi_2$ .

Let  $d_1$  be the syntactic degree of  $\Phi_1$  (and  $d - d_1$  be the syntactic degree of  $\Phi_2$ ). We distinguish between the following two subcases:

- **Case 2a:**  $i \leq md_1$ .

$$\begin{aligned}
& \sum_{t_{i+1}, \dots, t_{md} \in H} \pi^d(c_1, \dots, c_i, t_{i+1}, \dots, t_{md}) \cdot \hat{\Phi}(c_1, \dots, c_i, t_{i+1}, \dots, t_{md}) = \\
& \sum_{t_{i+1}, \dots, t_{md} \in H} \pi^d(c_1, \dots, c_i, t_{i+1}, \dots, t_{md}) \cdot \hat{\Phi}_1(c_1, \dots, c_i, t_{i+1}, \dots, t_{md_1}) \cdot \hat{\Phi}_2(t_{md_1+1}, \dots, t_{md}) = \\
& \left( \sum_{t_{i+1}, \dots, t_{md_1} \in H} \pi^{d_1}(c_1, \dots, c_i, t_{i+1}, \dots, t_{md_1}) \cdot \hat{\Phi}_1(c_1, \dots, c_i, t_{i+1}, \dots, t_{md_1}) \right) \cdot \\
& \left( \sum_{t_{md_1+1}, \dots, t_{md} \in H} \pi^{d-d_1}(t_{md_1+1}, \dots, t_{md}) \cdot \hat{\Phi}_2(t_{md_1+1}, \dots, t_{md}) \right)
\end{aligned}$$

– **Case 2b:**  $i > md_1$ .

$$\begin{aligned}
& \sum_{t_{i+1}, \dots, t_{md} \in H} \pi^d(c_1, \dots, c_i, t_{i+1}, \dots, t_{md}) \cdot \hat{\Phi}(c_1, \dots, c_i, t_{i+1}, \dots, t_{md}) = \\
& \sum_{t_{i+1}, \dots, t_{md} \in H} \pi^d(c_1, \dots, c_i, t_{i+1}, \dots, t_{md}) \cdot \hat{\Phi}_1(c_1, \dots, c_{md_1}) \cdot \hat{\Phi}_2(c_{md_1+1}, \dots, c_i, t_{i+1}, \dots, t_{md}) = \\
& \pi^{d_1}(c_1, \dots, c_{md_1}) \cdot \hat{\Phi}_1(c_1, \dots, c_{md_1}) \cdot \\
& \left( \sum_{t_{i+1}, \dots, t_{md} \in H} \pi^{d-d_1}(c_{md_1+1}, \dots, c_i, t_{i+1}, \dots, t_{md}) \cdot \hat{\Phi}_2(c_{md_1+1}, \dots, c_i, t_{i+1}, \dots, t_{md}) \right)
\end{aligned}$$

In both subcases  $\mathcal{M}(\Phi, \pi, i, c_1, \dots, c_i)$  computes Expression (3) recursively: It computes separately the part that corresponds to  $\hat{\Phi}_1$  and the part that corresponds to  $\hat{\Phi}_2$ , and then it multiplies the results.

■

**Theorem 1.** Fix parameters  $d, \mathbb{F}, H, m$ . Let  $\Psi : \mathbb{F}^k \rightarrow \mathbb{F}$  be an arithmetic formula of syntactic degree  $\leq d < |\mathbb{F}|$ . Let  $w = (w_1, \dots, w_k) \in \mathbb{F}^k$ , and let  $\pi = \text{LDE}_{\mathbb{F}, H, m}(1, w_1, \dots, w_k)$  be the low degree extension of  $(1, w_1, \dots, w_k)$  with respect to  $\mathbb{F}, H, m$ . Then the protocol  $(P_1(w, \Psi), V_1^\pi(\Psi))$  has the following guarantees.

- **Completeness:** If  $\Psi(w) = 0$  then

$$\Pr[(P_1(w, \Psi), V_1^\pi(\Psi)) = 1] = 1$$

- **Soundness:** If  $\Psi(w) \neq 0$  then for every (unbounded) interactive Turing machine  $\tilde{P}$ ,

$$\Pr[(\tilde{P}(w, \Psi), V_1^\pi(\Psi)) = 1] \leq \frac{(m \cdot |H| \cdot d)^2}{|\mathbb{F}|}$$

- **Complexity:** The prover and the verifier run in time  $\leq \text{poly}(|\Psi|, |\mathbb{F}|^m)$ . The verifier queries the oracle  $\pi$  at  $d$  points. The communication complexity is  $\leq \text{poly}(|\mathbb{F}|, m)$ .

Moreover,<sup>5</sup> the verifier can partition his work into two phases: In the first phase he runs in time  $\leq \text{poly}(|\Psi|, |\mathbb{F}|^m)$ , and generates a string  $h$  of size  $\leq \text{poly}(|\mathbb{F}|, m)$  of field elements. This phase depends only on  $\Psi$  and on the parameters (and on the randomness of the verifier) and does not depend on the oracle  $\pi$  or on the interaction with the prover. In the second phase, which is the interactive phase, the verifier uses only the string  $h$  generated in the first phase (and not the instance  $\Psi$ ), and runs in time  $\leq \text{poly}(|\mathbb{F}|, m)$ . The messages that he sends in this phase, and the oracle queries, depend only on  $h$  and are independent of the messages sent by the prover or the oracle answers.

**Proof of Theorem 1:** The completeness follows from Proposition 4.1, Claim 4.3, and Lemma 3.4. As for soundness, Proposition 3.1 implies that  $\pi$  is of degree  $< m|H|$ . Claim 4.2 implies that  $\hat{\Phi}$  is of degree  $\leq m|H|d$ . Thus,  $f_{\Phi, \pi}$  is of degree  $< 2m|H|d$ . Lemma 3.4 implies that for every (unbounded) interactive Turing machine  $\tilde{P}$ ,

$$\Pr[(\tilde{P}(w, \Psi), V_1^\pi(\Psi)) = 1] \leq \frac{md \cdot 2m|H|d}{|\mathbb{F}|} \leq \frac{(m \cdot |H| \cdot d)^2}{|\mathbb{F}|}.$$

As for the complexity: By Proposition 4.1, the verifier can compute  $\Phi$  in time  $\leq \text{poly}(d, |\Psi|)$ . By Claim 4.4, the verifier can compute  $\hat{\Phi}$  at one point in time  $\leq \text{poly}(d, |\Psi|, |\mathbb{F}|^m)$ . By Lemma 3.4, the verifier can run the sum-check procedure in time  $\leq \text{poly}(|\mathbb{F}|, m, d)$ , given an oracle query to  $\hat{\Phi}$  and oracle queries to  $\pi$ . In total, the verifier runs in time  $\leq \text{poly}(d, |\Psi|, |\mathbb{F}|^m) \leq \text{poly}(|\Psi|, |\mathbb{F}|^m)$ , and uses  $d$  oracle queries to  $\pi$ . These, together with Lemma 4.5, imply that the prover runs in time  $\leq \text{poly}(|\Psi|, |\mathbb{F}|^m)$ . From Lemma 3.4, the communication complexity is  $\leq \text{poly}(|\mathbb{F}|, m, d) \leq \text{poly}(|\mathbb{F}|, m)$ .

Moreover, the verifier can partition his work into two phases: In the first phase the verifier chooses randomly  $z^1, \dots, z^d \in_R \mathbb{F}^m$ , and computes  $\hat{\Phi}(z^1, \dots, z^d)$ . He lets

$$h \stackrel{\text{def}}{=} (z^1, \dots, z^d, \hat{\Phi}(z^1, \dots, z^d)).$$

Note that  $h$  is a string of size  $\leq \text{poly}(|\mathbb{F}|, m, d) \leq \text{poly}(|\mathbb{F}|, m)$ .

In the second phase, the verifier only runs the sum-check protocol  $(P_{\text{SC}}(f_{\Phi, \pi}), V_{\text{SC}}^{(f_{\Phi, \pi})})$  while using  $(z^1, \dots, z^d) \in \mathbb{F}^{md}$  as his random coin tosses. Thus, the messages that he sends during this phase:

$$(c_1, \dots, c_{md}) \stackrel{\text{def}}{=} (z^1, \dots, z^d),$$

are independent of the messages sent by the prover. The verifier then needs to query the oracle  $f_{\Phi, \pi}$  at the single point  $(z^1, \dots, z^d)$ . The verifier (who has oracle access to  $\pi$ ) computes

$$f_{\Phi, \pi}(z^1, \dots, z^d) \stackrel{\text{def}}{=} \pi(z^1) \cdot \dots \cdot \pi(z^d) \cdot \hat{\Phi}(z^1, \dots, z^d)$$

on his own. He queries the oracle  $\pi$  at the  $d$  points  $z^1, \dots, z^d \in \mathbb{F}^m$ , computes  $\pi(z^1) \cdot \dots \cdot \pi(z^d)$ , and multiplies it with  $\hat{\Phi}(z^1, \dots, z^d)$  (which he computed in the first phase). Lemma 3.4 implies that the running time of the verifier in the second phase is  $\leq \text{poly}(\mathbb{F}, m, d) = \text{poly}(\mathbb{F}, m)$ . ■

---

<sup>5</sup>This property is not needed for the main result. It is an additional feature that may be important for applications. We use it for the application of zero-knowledge.



## 5 Interactive PCP for Arithmetic Formulas: Part II

Fix a parameter  $d \in \mathbb{N}$ . In this section, we give an interactive PCP for proving the satisfiability of formulas of the form

$$\bigwedge_{i=1}^N [\Psi_i(x_1, \dots, x_k) = 0],$$

where  $\Psi_1(x_1, \dots, x_k), \dots, \Psi_N(x_1, \dots, x_k)$  are arithmetic formulas that compute polynomials of syntactic degree  $\leq d$ . We assume for simplicity that these formulas are over  $\mathbb{GF}[2]$ , though our results hold over any finite field. We construct an interactive PCP with the following parameters. Size of the PCP:  $p = \text{poly}(k, d)$ . Number of queries:  $q = \text{poly}(\log \log k, d)$ . Size of the interactive proof:  $\ell = \text{poly}(\log k, d, \log N)$ . Completeness:  $c = 1$ . Soundness:  $s < \frac{1}{2}$ . Moreover, the string  $\pi$  (generated by the prover in the first round of the protocol) depends only on the witness  $w_1, \dots, w_k$  (and on the parameter  $d$ ), and not on  $\Psi_1, \dots, \Psi_N$ . Actually,  $\pi$  will be the low degree extension with respect to some  $\mathbb{F}, H, m$ , specified below.

Formally, let  $\mathcal{E}_{k,d}$  be the class of all arithmetic formulas  $\Psi : \{0, 1\}^k \rightarrow \{0, 1\}$  of syntactic degree  $\leq d$ . Let

$$\text{SAT}_{N,k,d} = \left\{ (\Psi_1, \dots, \Psi_N) \in (\mathcal{E}_{k,d})^N : \left( \exists x \in \{0, 1\}^k \text{ s.t. } \bigwedge_{i=1}^N [\Psi_i(x) = 0] \right) \right\}.$$

We assume w.l.o.g. that  $|\Psi_1| + \dots + |\Psi_N| \geq k, d$ . Our main theorem is more general than stated above. Note that Theorem 2 is farther improved by Theorem 5 and Theorem 6 in Section 6.

**Theorem 2.** *For any soundness parameter<sup>6</sup>  $s \geq 2^{-n}$ ,*

$$\text{SAT}_{N,k,d} \in \text{IPCP}(p, q, \ell, c, s),$$

*with  $p = \text{poly}(k, d)$ ,  $q = \text{poly}(\log \log k, d, \log \frac{1}{s})$ ,  $\ell = \text{poly}(\log k, d, \log N, \log \frac{1}{s})$ , and  $c = 1$ .*

*Moreover, the following two properties can be attained.<sup>7</sup>*

1. *The string  $\pi$  (generated by the prover in the first round of the protocol) depends only on the witness  $w = (w_1, \dots, w_k)$  and on the parameter  $d$ , and not on the instance  $\Psi_1, \dots, \Psi_N$ .*
2. *The verifier can partition his work into two phases: In the first phase he runs in time  $\leq \text{poly}(|\Psi_1| + \dots + |\Psi_N|)$ , and generates a string  $h$  of size  $\leq \text{poly}(\log N, \log k, d, \log \frac{1}{s})$ . This phase depends only on  $\Psi_1, \dots, \Psi_N$  and on the parameters (and on the randomness of the verifier) and does not depend on the oracle  $\pi$  or on the interaction with the prover. In the second phase, which is the interactive phase, the verifier uses only the string  $h$  generated in the first phase (and not the instance  $\Psi_1, \dots, \Psi_N$ ), and runs in time  $\leq \text{poly}(\log N, \log k, d, \log \frac{1}{s})$ . The messages that he sends in this phase, and the oracle queries, depend only on  $h$  and are independent of the messages sent by the prover or the oracle answers.*

<sup>6</sup>We require  $s \geq 2^{-n}$  in order to ensure that the prover and verifier run in time polynomial in the size of the instance. We could take  $2^{-n} > s > 0$  and then the running time is polynomial in  $\log \frac{1}{s}$ .

<sup>7</sup>The second property is not needed for the main result. It is an additional feature that may be important for applications. We use it for the application of zero-knowledge.

## 5.1 Preliminaries

Fix parameters  $N, k, d \in \mathbb{N}$ . Before presenting our interactive PCP for  $\text{SAT}_{N,k,d}$ , we fix a few parameters and notation that will be used in our protocol. Assume for simplicity that  $d \cdot \log(k+1)$  is an integer which is a power of 2. This is without loss of generality, since it can be achieved by increasing  $k$  by at most a quadratic factor and increasing  $d$  by a constant factor. Increasing the parameter  $k$  can be done by adding dummy variables. Note that this does not change the guarantees in the statement of Theorem 2.

Let  $\mathbb{F}$  be an extension field of  $\mathbb{GF}[2]$  of size  $(d \cdot \log(k+1))^c$ , for some large enough constant integer  $c \in \mathbb{N}$  (to be determined below). Let  $H \subset \mathbb{F}$  be a subset of size  $d \cdot \log(k+1)$ . Let  $m = \left\lceil \frac{\log(k+1)}{\log|H|} \right\rceil$ . Assume for simplicity that  $|H|^m = k+1$ . This is without loss of generality since, once  $H$  and  $m$  are fixed, we can always add dummy variables to increase  $k$ . This will increase  $k$  by at most  $|H|$  and thus will not change the guarantees in the statement of Theorem 2. In order to make use of Lemma 3.2, we need to ensure that  $m \geq 3$ . To this end, we assume that  $k \geq d^4$ . As before, this is without loss of generality since we can always add  $d^4 - k$  dummy variables without changing the guarantees in the statement of Theorem 2. We also assume that  $k \geq 10$  and  $d \geq 15$ .

We fix the constant  $c \in \mathbb{N}$  so that the parameter  $\epsilon$  from Lemma 3.2 (in Subsection 3.2) satisfies

$$\epsilon \leq \frac{1}{d^2}, \quad (4)$$

and so that

$$\frac{(m|H|d)^2}{|\mathbb{F}|} \leq \frac{1}{d}. \quad (5)$$

Notice that since  $\mathbb{F}$  is an extension field of  $\mathbb{GF}[2]$ , we can view any arithmetic formula  $\Psi : \{0,1\}^k \rightarrow \{0,1\}$  (over  $\mathbb{GF}[2]$ ) as an arithmetic formula  $\Psi : \mathbb{F}^k \rightarrow \mathbb{F}$  (over  $\mathbb{F}$ ). Moreover, as in Section 4, we assume that addition and multiplication over  $\mathbb{F}$  are operations of complexity  $\leq \text{poly}(\log|\mathbb{F}|)$ .

## 5.2 The Protocol

In what follows, we describe the basic protocol that achieves soundness  $s \geq 1 - \frac{1}{d^2}$ . To improve the soundness parameter, we repeat this basic protocol sequentially.

**Parameters:**  $N, k, d, \mathbb{F}, H, m$  as described in Subsection 5.1.

**Input:** Both the prover and the verifier take as input  $N$  arithmetic formulas  $\Psi_1, \dots, \Psi_N : \{0,1\}^k \rightarrow \{0,1\}$  of syntactic degree  $\leq d$ . The prover takes an additional input  $w = (w_1, \dots, w_k) \in \{0,1\}^k$ , such that

$$\bigwedge_{i=1}^N [\Psi_i(w_1, \dots, w_k) = 0]. \quad (6)$$

### 1. Computing $\pi$ .

Let  $w_0 \stackrel{\text{def}}{=} 1$ . The prover generates

$$\pi \stackrel{\text{def}}{=} \text{LDE}_{\mathbb{F}, H, m}(w_0, w_1, \dots, w_k),$$

which is the low degree extension of  $(w_0, w_1, \dots, w_k)$  with respect to  $\mathbb{F}, H, m$ . The verifier is given oracle access to  $\pi$ .<sup>8</sup>

**Remarks:**

- (a) The fact that  $w_0 = 1$  implies that  $\pi(0^m) = 1$ .
- (b)  $\pi : \mathbb{F}^m \rightarrow \mathbb{F}$  is a multivariate polynomial of degree  $|H| - 1$  in each variable, and thus is of total degree  $\leq m \cdot (|H| - 1)$ .

**2. Running the low degree test on  $\pi$ .**

The verifier checks that  $\pi$  is close to an  $m$ -variate polynomial  $f : \mathbb{F}^m \rightarrow \mathbb{F}$  that has total degree  $\leq m \cdot (|H| - 1)$ . This is done by running the low degree test  $(P_{\text{LDT}}(\pi), V_{\text{LDT}}^\pi)$  described in Subsection 3.2. If the test fails then the verifier rejects.

**3. Running the point test on  $\pi$ .**

The verifier checks that if  $\pi$  is close to an  $m$ -variate polynomial  $f : \mathbb{F}^m \rightarrow \mathbb{F}$  then  $f(0^m) = 1$ . This is done by running the point test  $V_{\text{PT}}^\pi$  described in Subsection 3.3. If the test fails then the verifier rejects.

Note that so far the protocol depends only on  $w$  and  $d$ , and does not depend on  $\Psi_1, \dots, \Psi_N$ .

**4. Restricting all satisfying assignments to bit strings.**

In our protocol, the prover proves that Equation (6) holds for the witness  $w$  encoded by  $\pi$ . In order to ensure soundness, the verifier should verify that  $w \in \{0, 1\}^k$ . (Note that it may be the case that  $\bigwedge_{i=1}^N [\Psi_i(w) = 0]$  does not have any satisfying assignment in  $\{0, 1\}^k$ , and yet there exists  $w \in \mathbb{F}^k$  that does satisfy  $\bigwedge_{i=1}^N [\Psi_i(w) = 0]$ ).

From now on, we think of  $\Psi_1, \dots, \Psi_N : \mathbb{F}^k \rightarrow \mathbb{F}$ . To ensure that  $w \in \{0, 1\}^k$ , we consider  $k$  additional arithmetic formulas

$$\Psi_{N+1}, \dots, \Psi_{N+k} : \mathbb{F}^k \rightarrow \mathbb{F}.$$

For every  $i \in [k]$ , the formula  $\Psi_{N+i}$  is defined by

$$\Psi_{N+i}(x_1, \dots, x_k) \stackrel{\text{def}}{=} x_i^2 - x_i.$$

The prover, instead of proving Equation (6), will prove that

$$\bigwedge_{i=1}^{N+k} [\Psi_i(w_1, \dots, w_k) = 0]. \tag{7}$$

Equation (7) implies in particular that  $w_1, \dots, w_k \in \{0, 1\}$ .

**5. Applying an error correcting code.**

Next, we reduce the task of proving the satisfiability of a conjunction of  $N + k$  arithmetic formulas of syntactic degree  $\leq d$  over  $\mathbb{F}$ , to the task of proving the satisfiability of a *single*

---

<sup>8</sup>As opposed to the protocol  $(P_1(w, \Psi), V_1^\pi(\Psi))$  described in Section 4, here if the prover is dishonest then  $\pi$  is arbitrary.

arithmetic formula of syntactic degree  $\leq d$  over  $\mathbb{F}$ . For this we use a linear error correcting code

$$ECC : \mathbb{F}^{N+k} \rightarrow \mathbb{F}^M,$$

with relative distance  $\frac{1}{3}$  and with  $M = O(N+k)$ . The verifier chooses a random coordinate  $j \in_R [M]$  and sends it to the prover. Let  $\Psi : \mathbb{F}^k \rightarrow \mathbb{F}$  be the arithmetic formula of syntactic degree  $\leq d$  defined by

$$\Psi(x) \stackrel{\text{def}}{=} ECC(\Psi_1(x), \dots, \Psi_{N+k}(x))_j.$$

## 6. Checking that $\Psi(w) = 0$ .

For this, the prover and the verifier run the protocol  $(P_1(w, \Psi), V_1^\pi(\Psi))$  described in Section 4.

We denote the protocol above by  $(P_2(w, \Psi_1, \dots, \Psi_N), V_2(\Psi_1, \dots, \Psi_N))$ . A succinct description of it appears in Figure 5.

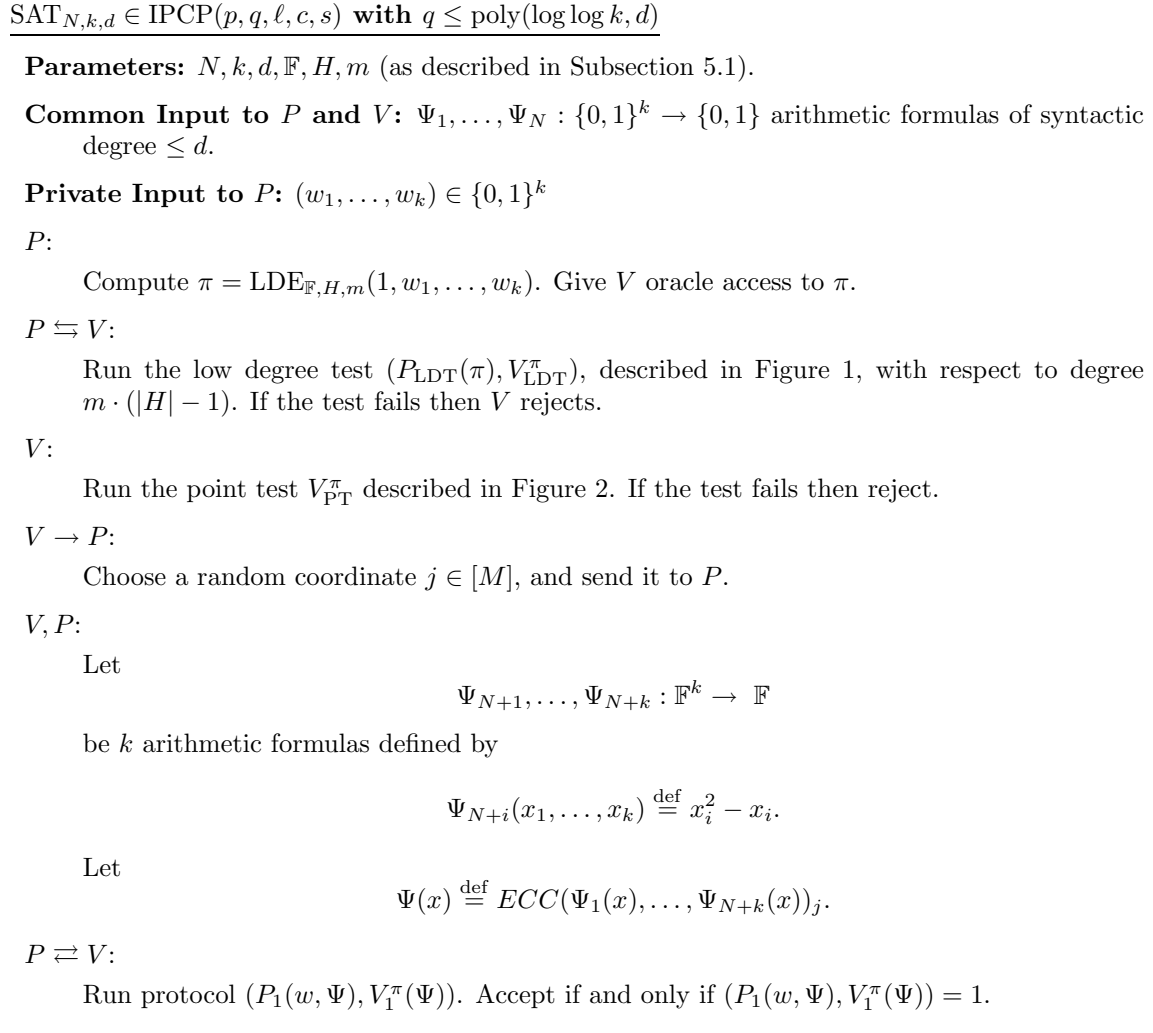


Figure 5:  $(P_2(w, \Psi_1, \dots, \Psi_N), V_2(\Psi_1, \dots, \Psi_N))$

**Theorem 3.** *The protocol described in Figure 5 is an interactive PCP for  $\text{SAT}_{N,k,d}$  with parameters  $(p, q, \ell, c, s)$ , where  $p \leq \text{poly}(k, d)$ ,  $q \leq \text{poly}(\log \log k, d)$ ,  $\ell \leq \text{poly}(\log k, d, \log N)$ ,  $c = 1$ , and  $s \leq 1 - \frac{1}{d^2}$ .*

*Moreover, the protocol satisfies the following two properties.<sup>9</sup>*

1. *The string  $\pi$  (generated by the prover in the first round of the protocol) depends only on the witness  $w = (w_1, \dots, w_k)$  and on the parameter  $d$ , and not on the instance  $\Psi_1, \dots, \Psi_N$ .*
2. *The verifier can partition his work into two phases: In the first phase he runs in time  $\leq \text{poly}(|\Psi_1| + \dots + |\Psi_N|)$ , and generates a string  $h$  of size  $\leq \text{poly}(\log N, \log k, d)$ . This phase depends only on  $\Psi_1, \dots, \Psi_N$  and on the parameters (and on the randomness of the verifier) and does not depend on the oracle  $\pi$  or on the interaction with the prover. In the second phase, which is the interactive phase, the verifier uses only the string  $h$  generated in the first phase (and not the instance  $\Psi_1, \dots, \Psi_N$ ), and runs in time  $\leq \text{poly}(\log N, \log k, d)$ . The messages that he sends in this phase, and the oracle queries, depend only on  $h$  and are independent of the messages sent by the prover or the oracle answers.*

**Proof of Theorem 3:** We first note that our choice of parameters implies that  $|\mathbb{F}|^m = \text{poly}(k, d)$ . Lemma 3.2, Lemma 3.3, and Theorem 1 imply that both the prover  $P$  and the verifier  $V$  run in time  $\leq \text{poly}(|\mathbb{F}|^m, |\Psi_1| + \dots + |\Psi_N|)$ , and recall that we assume w.l.o.g. that  $|\Psi_1| + \dots + |\Psi_N| \geq k, d$ . Thus,  $P$  and  $V$  are indeed (probabilistic) polynomial time Turing machines.

Proposition 3.1 implies that the length of  $\pi$  is  $p \leq \text{poly}(|\mathbb{F}|^m) = \text{poly}(k, d)$ . Lemma 3.2 and Theorem 1 (together with the sending of the index  $j \in [M]$ ) imply that the communication complexity is  $\ell \leq \text{poly}(|\mathbb{F}|, m, \log M) = \text{poly}(\log k, d, \log N)$ . The verifier  $V$  sends  $2d + 2$  oracle queries to  $\pi$ : 1 during the low degree test,  $d + 1$  during the point test, and  $d$  during the execution of  $(P_1(w, \Psi), V_1(\Psi))$ . Note, however, that each query returns a field element, and thus corresponds to  $O(\log d + \log \log k)$  bits. thus,  $q \leq \text{poly}(\log \log k, d)$ . Lemma 3.2, Lemma 3.3, and Theorem 1 imply that our completeness parameter is  $c = 1$ .

We next show that the soundness parameter is  $s \leq 1 - \frac{1}{d^2}$ . Fix any  $(\Psi_1, \dots, \Psi_N) \notin \text{SAT}_{N,k,d}$ , any unbounded (cheating) prover  $\tilde{P}$ , and any function  $\tilde{\pi} : \mathbb{F}^m \rightarrow \mathbb{F}$ . Define

$$(\tilde{w}_0, \tilde{w}_1, \dots, \tilde{w}_k) \in \mathbb{F}^{k+1}$$

by  $\tilde{w}_i \stackrel{\text{def}}{=} \tilde{\pi}(\alpha^{-1}(i))$ , where  $\alpha : H^m \rightarrow \{0, 1, \dots, k\}$  is the lexicographic order of  $H^m$ . Let  $S$  denote the event that  $(\tilde{P}(\Psi_1, \dots, \Psi_N), V^{\tilde{\pi}}(\Psi_1, \dots, \Psi_N)) = 1$ , and let  $s \stackrel{\text{def}}{=} \Pr[S]$ . Assume for the sake of contradiction that

$$s > 1 - \frac{1}{d^2}. \tag{8}$$

According to Lemma 3.2, there exists an  $m$ -variate polynomial  $f$  over  $\mathbb{F}$  of degree  $\leq m \cdot (|H| - 1)$  such that

$$\Pr_{z \in_R \mathbb{F}^m} [\tilde{\pi}(z) = f(z)] \geq s - \epsilon,$$

where  $\epsilon$  is defined in Lemma 3.2. Let

$$\gamma \stackrel{\text{def}}{=} 1 - (s - \epsilon).$$

---

<sup>9</sup>The second property is not needed for the main result. It is an additional feature that may be important for applications. We use it for the application of zero-knowledge.

Equation (4) and Equation (8) imply that

$$\gamma \leq \frac{2}{d^2}. \quad (9)$$

We next show that it must be the case that  $f(0^m) = 1$ . The reason is that if  $f(0^m) \neq 1$ , then Lemma 3.3, together with the fact that  $|\mathbb{F}|^m \geq d^2$ , implies that

$$s \leq \Pr[V_{\text{PT}}^{\tilde{\pi}} = 1] \leq (d+1) \left( \gamma + \frac{1}{|\mathbb{F}|^m} \right) \leq (d+1) \left( \frac{2}{d^2} + \frac{1}{d^2} \right) = \frac{3(d+1)}{d^2} < 1 - \frac{1}{d^2}.$$

This contradicts Equation (8).

Let  $A$  denote the event that the coordinate  $j \in [M]$  chosen by  $V$  satisfies

$$\Psi(\tilde{w}) \stackrel{\text{def}}{=} ECC(\Psi_1(\tilde{w}), \dots, \Psi_{N+k}(\tilde{w}))_j \neq 0.$$

Then,

$$\Pr[A] \geq \frac{1}{3}. \quad (10)$$

Recall that when running the protocol  $(P_1(w, \Psi), V_1^{\tilde{\pi}}(\Psi))$ , the verifier  $V$  queries the oracle at  $d$  points. Let  $B$  denote the event that on these  $d$  points  $\tilde{\pi}$  is consistent with  $f$ . Note that

$$\Pr[\neg(B)] \leq d\gamma \leq \frac{2}{d}. \quad (11)$$

From Bayes rule

$$s = \Pr[S] \leq \Pr[S|A \wedge B] + \Pr[\neg(A) \vee \neg(B)].$$

According to Theorem 1,

$$\Pr[S|A \wedge B] \leq \frac{(m \cdot |H| \cdot d)^2}{|\mathbb{F}|}.$$

This, together with Equation (5), implies that

$$\Pr[S|A \wedge B] \leq \frac{1}{d}.$$

The union bound, together with Equations (10) and (11), implies that

$$\Pr[\neg(A) \vee \neg(B)] \leq \frac{2}{3} + \frac{2}{d}.$$

Thus, all in all we have

$$s \leq \frac{3}{d} + \frac{2}{3} < 1 - \frac{1}{d^2},$$

contradicting Equation (8).

It remains to show that the two additional properties, required by the statement in Theorem 3, are attained.

1. The fact that  $\pi$  depends only on the witness  $w = (w_1, \dots, w_k)$  and on the parameter  $d$ , follows immediately from the definition of  $\pi \stackrel{\text{def}}{=} \text{LDE}_{\mathbb{F}, H, m}(1, w_1, \dots, w_k)$  and from the fact that the parameters  $\mathbb{F}, H, m$  depend only on  $k$  and  $d$ .

2. The verifier can partition his work into two phases. In the first phase he does the following:

- (a) He computes  $\Psi_{N+1}, \dots, \Psi_{N+k}$ , as in step 4.
- (b) He chooses a random coordinate  $j \in_R [M]$ , and computes

$$\Psi(x) \stackrel{\text{def}}{=} ECC(\Psi_1(x), \dots, \Psi_{N+k}(x))_j,$$

as in step 5.

- (c) Recall that according to Theorem 1, the verifier,  $V_1^\pi(\Psi)$  in the protocol  $(P_1(w, \Psi), V_1^\pi(\Psi))$  can also partition his work into two phases. The verifier computes  $h_1$ , which is the string computed by  $V_1(\Psi)$  in the first phase of the protocol  $(P_1(w, \Psi), V_1^\pi(\Psi))$ .
- (d) He sets  $h \stackrel{\text{def}}{=} (j, h_1)$ .

Note that the running time of the verifier in the first phase is  $\leq \text{poly}(|\Psi_1| + \dots + |\Psi_N|)$ . According to Theorem 1 the size of  $h_1$  is  $\leq \text{poly}(|\mathbb{F}|, m) = \text{poly}(\log k, d)$ , and thus the size of  $h$  is  $\leq \text{poly}(\log N, \log k, d)$ .

In the second phase the verifier does the following:

- (a) He runs the low degree test on  $\pi$ , as in step 2.
- (b) He runs the point test on  $\pi$ , as in step 3.
- (c) If both tests pass, then he sends the coordinate  $j \in [M]$  (computed in the first phase) to the prover, and runs the second phase of  $(P_1(w, \Psi), V_1^\pi(\Psi))$ , using  $h_1$  (computed in the first phase).

Lemma 3.2, Lemma 3.3 and Theorem 1 (together with the sending of  $j \in [M]$ ) imply that the running time of the verifier in this phase is  $\leq \text{poly}(\log M, |\mathbb{F}|, m) \leq \text{poly}(\log N, \log k, d)$ , and that the messages that he sends are independent of the messages sent by the prover.

■

**Proof of Theorem 2:** Theorem 2 follows as a corollary of Theorem 3, by repeating steps 2-6, in the protocol described above, sequentially  $\text{poly}(d, \log \frac{1}{s})$  times.

■

## 6 Reducing the Number of Queries

In this section, we show how to reduce the number of queries in an interactive PCP to one. We first prove a general theorem that shows that in any interactive PCP the number of queries can be reduced to one, with a small payment in the other parameters.

**Theorem 4.** *If  $L \in \text{IPCP}(p, q, \ell, c, s)$  then for any<sup>10</sup>  $\epsilon \geq 1/n$ ,  $L \in \text{IPCP}(p', q', \ell', c', s')$ , with  $p' = \text{poly}(p/\epsilon)$ ,  $q' = 1$ ,  $\ell' = \text{poly}(\ell, q, \log p, 1/\epsilon)$ ,  $c' = c$ , and  $s' = 1/2 + O(\sqrt[6]{s} + \epsilon)$ .*

*Moreover, if  $(P_0, V_0)$  is an interactive PCP with parameters  $(p, q, \ell, c, s)$  for  $L$ , there exists an interactive PCP  $(P, V)$  with parameters  $(p', q', \ell', c', s')$  for  $L$ , s.t.:*

---

<sup>10</sup>We require  $\epsilon \geq 1/n$  in order to ensure that the prover and verifier run in time polynomial in the size of the instance. We could take  $1/n > \epsilon > 0$  and then the running time is polynomial in  $1/\epsilon$

1. The string  $\pi$ , generated by  $P$ , depends only on the string  $\pi_0$ , generated by  $P_0$ , and on the parameters  $q, \epsilon$ , and not on the instance  $x$ .
2. Assume that  $V_0$  can partition his work into two phases: In the first phase he runs in time  $T_1$ , and generates a string  $g$  of size  $S_1$ . This phase depends only on  $x$  and on the parameters (and on the randomness of  $V_0$ ) and does not depend on  $\pi_0$  or on the interaction with  $P_0$ . In the second phase, which is the interactive phase,  $V_0$  uses only the string  $g$  generated in the first phase (and not the instance  $x$ ), and runs in time  $T_2$ . The messages that he sends in this phase, and the oracle queries, depend only on  $g$  (and the randomness of  $V_0$ ) and are independent of the messages sent by the prover or the oracle answers.

Then  $V$  can also partition his work into two phases, with the exact same properties and parameters, except that the running time of the second phase is  $\text{poly}(T_2, q, \log p, 1/\epsilon)$ .

**Proof of Theorem 4:** Assume w.l.o.g. that  $p \geq q$  (there is no reason to query more than all the points in a string). Let  $h$  be the smallest integer that is a power of 2 and such that

$$h \geq \max\{q, \log p, 1/\epsilon, 2\}.$$

Assume w.l.o.g. that

$$p \geq h.$$

Otherwise, we just increase  $p$  (to be  $h$ , which is at most  $\max\{2p, 2/\epsilon, 2\}$ ), and note that this doesn't effect the parameters  $(p', q', \ell', c', s')$  in the statement of the theorem. Assume w.l.o.g. that

$$\ell \geq h.$$

Otherwise, we just increase  $\ell$  (to be  $h$ ) and once again this doesn't effect the parameters  $(p', q', \ell', c', s')$  in the statement of the theorem.

Let  $(P_0, V_0)$  be an interactive PCP with parameters  $(p, q, \ell, c, s)$  for  $L$ . Using the above mentioned assumptions, we just need to show an interactive PCP for  $L$ , with parameters  $(p', q', \ell', c', s')$  with  $p' = \text{poly}(p)$ ,  $q' = 1$ ,  $\ell' = \text{poly}(\ell)$ ,  $c' = c$ , and  $s' = 1/2 + O(\sqrt[6]{s} + \epsilon)$ .

Let  $\pi_0$  be the bit string of size at most  $p$ , generated by the prover  $P_0$ . Without loss of generality, we assume that all the queries made by the verifier  $V_0$  to the string  $\pi_0$  are made after the end of the interaction between  $P_0$  and  $V_0$ . This can be assumed because rather than querying  $\pi_0$  in the middle of the interaction, the verifier can simply ask the prover to supply the answers, and after the interaction ends the verifier can make the actual queries to  $\pi_0$  and reject if the prover cheated. This increases  $\ell$  by at most  $\text{poly}(q, \log p) \leq \text{poly}(\ell)$ , and doesn't effect any of the other parameters.

We define  $\pi_1$  to be the low degree extension of  $\pi_0$  as follows. Recall that  $h$  is a power of 2. Let  $\mathbb{F}$  be an extension field of  $\mathbb{GF}[2]$  of size  $h^c$ , for some large enough constant integer  $c \in \mathbb{N}$  (to be determined later on). Let  $H \subset \mathbb{F}$  be a subset of size  $h$ . Let  $m = \max\left\{\left\lceil \frac{\log p}{\log h} \right\rceil, 4\right\}$ . (In most interesting cases,  $m$  will be larger than a constant). Note that  $p \leq |H|^m \leq \text{poly}(p)$ . Define  $\pi'_0$  to be  $\pi_0$  padded by  $|H|^m - p$  arbitrary elements (say, zeros), and define  $\pi_1 = \text{LDE}_{\mathbb{F}, H, m}(\pi'_0)$ . By the definition of low degree extension (Subsection 3.1),  $\pi_1 : \mathbb{F}^m \rightarrow \mathbb{F}$  is a polynomial of total degree  $< mh$ , and there is a mapping  $E : [p] \rightarrow \mathbb{F}^m$  such that for every  $Q \in [p]$  we have  $\pi_0(Q) = \pi_1(E(Q))$ . (We think of  $\pi_0$  as a function from  $[p]$  to  $\{0, 1\}$ ).

The main step in the proof of the theorem will be to give an interactive PCP  $(P_1, V_1)$  that works with parameters  $(p_1, q_1, \ell_1, c_1, s_1)$ , such that  $p_1 = \text{poly}(p)$ ,  $q_1 = 1$ ,  $\ell_1 = \text{poly}(\ell)$ ,  $c_1 = c$ , as required, except that the one query made by  $V_1$  is to  $\pi_1$  and hence the query returns a field element in  $\mathbb{F}$ ,



rather than a single bit. (We can think of  $(P_1, V_1)$  as an interactive PCP with a larger answer size). The soundness parameter  $s_1$  will be at most  $O(\sqrt{s} + \epsilon)$ . (This is better than required in the statement of the theorem and is possible because  $\pi_1$  has answer size larger than 1).

### Description of $(P_1, V_1)$

The main idea of the construction is that if the verifier needs to query  $q$  points in  $\mathbb{F}^m$ , he takes a 4-dimensional manifold that contains all these points and a random 3-dimensional subspace, and asks the prover to give the restriction of  $\pi_1$  to this manifold. The verifier checks the answer by querying  $\pi_1$  in a single random point on the manifold. This tests both that  $\pi_1$  is close to a low degree polynomial and that the answers given by the prover agree with this polynomial. Since the manifold contains all the needed queries, the verifier can use the answers that were given by the prover, rather than querying  $\pi_1$ . To obtain small soundness, we will need to use the list-decoding soundness property of a low error low degree test (see Lemma 3.2).

We will now describe the interactive PCP  $(P_1, V_1)$ .

1. **Generating the string:**  $P_1$  generates the string of field elements  $\pi_1 = \text{LDE}_{\mathbb{F}, H, m}(\pi'_0) : \mathbb{F}^m \rightarrow \mathbb{F}$ , as defined above.
2. **Simulating the interaction:**  $(P_1, V_1)$  simulate the interaction between  $P_0$  and  $V_0$ . That is,  $P_1$  acts the same as  $P_0$  and  $V_1$  the same as  $V_0$ , until the interaction between  $P_0$  and  $V_0$  ends. Note that by our assumption,  $V_0$  still doesn't make any query to  $\pi_0$ .
3. **Generating the queries:** After the interaction ends,  $V_0$  wants to query  $\pi_0$  in points  $Q_1, \dots, Q_q \in [p]$ . Denote by  $x_1 = E(Q_1), \dots, x_q = E(Q_q)$  the corresponding points in  $\mathbb{F}^m$ . Recall that by the definition of  $\pi_1$ , for every  $i \in [q]$ , we have  $\pi_1(x_i) = \pi_0(Q_i)$ .
4. **A curve through the queries:**  $V_1$  chooses at random an additional element  $x_{q+1} \in_R \mathbb{F}^m$ . Denote by  $\gamma : \mathbb{F} \rightarrow \mathbb{F}^m$  a curve of degree at most  $q$  through  $x_1, \dots, x_{q+1}$ . Formally, we choose  $q+1$  distinct elements  $t_1, \dots, t_{q+1} \in \mathbb{F}$  (where, it will be convenient to assume that  $t_1, \dots, t_q$  are fixed and  $t_{q+1}$  is a random element in  $\mathbb{F} \setminus \{t_1, \dots, t_q\}$ ) and then  $\gamma : \mathbb{F} \rightarrow \mathbb{F}^m$  is a polynomial of degree at most  $q$  (i.e., each of its coordinates is a polynomial of degree at most  $q$  from  $\mathbb{F}$  to  $\mathbb{F}$ ), such that, for every  $i \in [q+1]$ , we have  $\gamma(t_i) = x_i$ . It is well known that for any  $t_1, \dots, t_{q+1}, x_1, \dots, x_{q+1}$ , these conditions determine  $\gamma$  uniquely and that  $\gamma$  can be computed by polynomial interpolation. It is well known that for any fixed  $t_1, \dots, t_{q+1}, x_1, \dots, x_q$ , if  $x_{q+1} \in_R \mathbb{F}^m$  then for every  $t \in \mathbb{F} \setminus \{t_1, \dots, t_q\}$ , the point  $\gamma(t)$  is a random variable uniformly distributed in  $\mathbb{F}^m$ .
5. **A manifold through the curve:**  $V_1$  chooses  $z_2, z_3 \in_R \mathbb{F}^m$ . Denote by  $\Gamma : \mathbb{F}^4 \rightarrow \mathbb{F}^m$  the 4 dimensional manifold defined by  $\Gamma(t, \alpha_1, \alpha_2, \alpha_3) = \alpha_1 \gamma(t) + \alpha_2 z_2 + \alpha_3 z_3$ . We think of  $\Gamma$  as the manifold spanned by both  $\gamma$  and the vector space spanned by  $z_2, z_3$ .
6. **Getting the values on the manifold:**  $V_1$  sends to  $P_1$  the set of all points in the image of  $\Gamma$  (at most  $|\mathbb{F}|^4 = \text{poly}(h)$  points).  $P_1$  answers by values that are supposed to be the value of  $\pi_1$  on all these points. Based on these values,  $V_1$  creates a function  $\rho : \mathbb{F}^4 \rightarrow \mathbb{F}$ , supposed to be the restriction of  $\pi_1$  to the image of  $\Gamma$ . Namely,  $\rho \stackrel{\text{def}}{=} \pi_1 \circ \Gamma : \mathbb{F}^4 \rightarrow \mathbb{F}$ . (We think of  $\rho$  as the answers given by the prover, reorganized as a function from  $\mathbb{F}^4$  to  $\mathbb{F}$ ).  $V_1$  checks that  $\rho : \mathbb{F}^4 \rightarrow \mathbb{F}$  is a polynomial of total degree at most  $m \cdot h \cdot (q+1)$ . (Recall that the degree of  $\pi_1$  is at most  $m \cdot h$  and the degree of  $\gamma$  is at most  $q$ , and hence the degree of  $\Gamma$  is at most  $q+1$

and the degree of  $\pi_1 \circ \Gamma$  is at most  $m \cdot h \cdot (q + 1)$ ). If  $\rho$  is not a polynomial of total degree at most  $m \cdot h \cdot (q + 1)$ , it is clear that the prover is cheating and  $V_1$  rejects.

7. **Low degree testing:**  $V_1$  chooses a random  $t \in_R \mathbb{F} \setminus \{t_1, \dots, t_q\}$  and computes  $z_1 = \gamma(t)$ . If  $z_1, z_2, z_3$  are linearly dependant  $V_1$  accepts. Otherwise, define  $\eta : \mathbb{F}^3 \rightarrow \mathbb{F}$  by  $\eta(\alpha_1, \alpha_2, \alpha_3) = \rho(t, \alpha_1, \alpha_2, \alpha_3)$ . The verifier checks that  $\eta$  is a polynomial of total degree at most  $m \cdot h$  (as it is supposed to be the restriction of  $\pi_1$  to a 3-dimensional subspace). If  $\eta$  is not a polynomial of total degree at most  $m \cdot h$ , it is clear that the prover is cheating and  $V_1$  rejects. Otherwise,  $V_1$  chooses random  $\alpha_1, \alpha_2, \alpha_3 \in_R \mathbb{F}$  and computes  $z = \alpha_1 z_1 + \alpha_2 z_2 + \alpha_3 z_3$ .  $V_1$  then queries  $\pi_1$  at the single point  $z$  and compares the result to the value given by the prover, that is, to  $\eta(\alpha_1, \alpha_2, \alpha_3)$ . If the answers are different it is clear that the prover is cheating and  $V_1$  rejects. (Note that if the prover is not cheating then  $\eta(\alpha_1, \alpha_2, \alpha_3) = \rho(t, \alpha_1, \alpha_2, \alpha_3) = \pi_1(\Gamma(t, \alpha_1, \alpha_2, \alpha_3)) = \pi_1(\alpha_1 \gamma(t) + \alpha_2 z_2 + \alpha_3 z_3) = \pi_1(\alpha_1 z_1 + \alpha_2 z_2 + \alpha_3 z_3) = \pi_1(z)$ ).
8. **Simulating the verifier:** Denote by  $a_1, \dots, a_q$  the answers that the prover gave on the points  $x_1, \dots, x_q$ . That is, for  $i \in [q]$ , denote  $a_i = \rho(t_i, 1, 0, 0)$ . (Note that if the prover is not cheating  $a_i = \rho(t_i, 1, 0, 0) = \pi_1(\Gamma(t_i, 1, 0, 0)) = \pi_1(\gamma(t_i)) = \pi_1(x_i)$ ). If for some  $i$ , the field element  $a_i$  is not in  $\{0, 1\}$  then  $V_1$  rejects. Otherwise,  $V_1$  simulates  $V_0$  by giving  $V_0$  the answers  $a_1, \dots, a_q$  for the queries  $Q_1, \dots, Q_q$  and accepts iff  $V_0$  accepts on these answers.

### Analysis of $(P_1, V_1)$

By the description of the protocol  $(P_1, V_1)$  and the assumptions on the parameters, it is straightforward to verify that the size of the bit description of  $\pi_1$  is  $\text{poly}(p)$  and the communication complexity between  $P_1$  and  $V_1$  is  $\text{poly}(\ell)$ .  $V_1$  queries  $\pi_1$  in a single point (and gets a field element as an answer). Also, it is straightforward to verify that if  $P_1, V_1$  act according to the protocol,  $V_1$  always accepts if the simulated verifier  $V_0$  accepts. Hence, the protocol has completeness  $\geq c$ . We will now prove the soundness property of the protocol.

### Soundness of $(P_1, V_1)$

Assume that the instance  $x$  is not in  $L$ . Let  $\tilde{P}_1$  be any (cheating) prover and let  $\tilde{\pi}_1$ , be any string. We will bound the probability for acceptance,

$$\Pr[(\tilde{P}_1(x), V_1^{\tilde{\pi}_1}(x)) = 1].$$

Let  $\delta = \max\{\sqrt{s}, 1/h\}$ , and let  $r = 2/\delta = \min\{2h, 2/\sqrt{s}\}$ . Let  $f_1, \dots, f_r : \mathbb{F}^m \rightarrow \mathbb{F}$  be the  $r$  polynomials of degree at most  $m \cdot h$  guaranteed for the function  $\tilde{\pi}_1 : \mathbb{F}^m \rightarrow \mathbb{F}$  by the list-decoding soundness condition of the low degree test (see Lemma 3.2), with degree  $m \cdot h$  and parameter  $\delta$ . Hence, by the list-decoding soundness property of the low degree test, for any (cheating) interactive Turing machine  $\tilde{P}_{LDT}$ ,

$$\Pr \left[ [(\tilde{P}_{LDT}(\tilde{\pi}_1), V_{LDT}^{\tilde{\pi}_1}) = 1] \wedge [\tilde{\pi}_1(z) \notin \{f_1(z), \dots, f_r(z)\}] \right] \leq O(\delta),$$

where  $z \in \mathbb{F}^m$  is the random element chosen in Step 3 of the low degree test.

(Note that the size of the field  $\mathbb{F}$  was chosen to be  $h^c$  for a large enough  $c$ , and we can take  $c$  to be a large enough constant so that  $\delta$  is larger than the value of  $\epsilon$  from Lemma 3.2).

Denote the following events:

1. Let  $E_0$  be the event that  $[(\tilde{P}_1(x), V_1^{\tilde{\pi}_1}(x)) = 1]$ , i.e., the protocol accepts.

2. Let  $E_1$  be the event that  $\forall i \in [r] : (a_1, \dots, a_q) \neq (f_i(x_1), \dots, f_i(x_q))$ , where  $x_1, \dots, x_q$  are defined in Step 3 of the protocol and  $a_1, \dots, a_q$  are the answers obtained in Step 8 of the protocol.
3. Let  $E_2$  be the event that  $\forall i \in [r] : \tilde{\pi}_1(z) \neq f_i(z)$ , where  $z$  is the point chosen in Step 7 of the protocol.

Denote by  $\tilde{\rho} : \mathbb{F}^4 \rightarrow \mathbb{F}$  the (reorganized) answer of the (cheating) prover  $\tilde{P}_1$  in Step 6 of the protocol. Assume w.l.o.g. that  $\tilde{\rho}$  is always a polynomial of degree at most  $m \cdot h \cdot (q+1)$  (as otherwise the verifier rejects). Denote by  $\tilde{\eta} : \mathbb{F}^3 \rightarrow \mathbb{F}$  the function  $\tilde{\eta}(\alpha_1, \alpha_2, \alpha_3) = \tilde{\rho}(t, \alpha_1, \alpha_2, \alpha_3)$ , where  $t$  is the element chosen in Step 7 of the protocol. Assume w.l.o.g. that  $\tilde{\eta}$  is always a polynomial of degree at most  $m \cdot h$  (as otherwise the verifier rejects). Assume for simplicity that  $z_1, z_2, z_3$ , generated in Step 7 of the protocol are not linearly dependent, as this occurs with a negligible probability  $\leq O(\delta)$  and hence doesn't effect the correctness of the following three claims. Also, it will be convenient to assume that if the protocol rejects in Step 7 it still continues to Step 8 (although the verifier already rejected). This is convenient to assume because otherwise the variables defined in Step 8 are not always defined and hence the event  $E_1$  is not well defined.

The soundness property of  $(P_1, V_1)$  follows easily by the following three claims.

**Claim 6.1.**

$$\Pr[E_2 \wedge E_0] \leq O(\delta).$$

**Proof of Claim 6.1:** The proof will follow from the above mentioned list-decoding soundness property of the low degree test, described in Subsection 3.2 (see Lemma 3.2).

Note that the elements  $z_1, z_2, z_3 \in \mathbb{F}^m$ , chosen in Step 5 and Step 7 of the protocol, are independent and uniformly distributed random variables as in the low degree test. We think of the function  $\tilde{\eta} : \mathbb{F}^3 \rightarrow \mathbb{F}$  as an answer given by a (possibly cheating) prover in the low degree test. In Step 7 of the protocol the verifier checks that  $\tilde{\eta}$  is a low degree polynomial and checks that  $\tilde{\eta}(\alpha_1, \alpha_2, \alpha_3) = \tilde{\pi}_1(\alpha_1 z_1 + \alpha_2 z_2 + \alpha_3 z_3)$ , where  $\alpha_1, \alpha_2, \alpha_3$  are random elements in  $\mathbb{F}^m$ . Note that this is an exact implementation of the low degree test of Subsection 3.2.

Formally, given the verifier  $V_1$  and a prover  $\tilde{P}_1$ , a prover  $\tilde{P}_{LDT}$  for the low degree test can simulate the interaction between  $V_1$  and  $\tilde{P}_1$  as follows:  $\tilde{P}_{LDT}$  gets  $z_1, z_2, z_3$  from the verifier  $V_{LDT}$  of the low degree test.  $\tilde{P}_{LDT}$  simulates Step 2 - Step 7 of the interaction between  $V_1$  and  $\tilde{P}_1$  using  $z_1$  (that was given by  $V_{LDT}$ ) instead of  $x_{q+1}$  in Step 4 and using  $z_1, z_2, z_3$  that were given by  $V_{LDT}$  in Step 5 and Step 7, and using  $t = t_{q+1}$  in Step 7.  $\tilde{P}_{LDT}$  gives the function  $\tilde{\eta}$  as an answer to  $V_{LDT}$ . Note that if  $V_{LDT}$  rejects, the protocol  $(P_1, V_1)$  rejects as well.

By the definition of the polynomials  $f_1, \dots, f_r$ , and by Lemma 3.2, we conclude that the probability that the verifier accepts and  $\tilde{\pi}_1(z) \notin \{f_1(z), \dots, f_r(z)\}$  is  $\leq O(\delta)$ .

(Note that the size of the field  $\mathbb{F}$  was chosen to be  $h^c$  for a large enough  $c$ , and we can take  $c$  to be a large enough constant so that  $\delta$  is larger than the value of  $\epsilon$  from Lemma 3.2). ■

**Claim 6.2.**

$$\Pr[\neg E_2 \wedge E_1 \wedge E_0] \leq O(\delta).$$

**Proof of Claim 6.2:** The proof will follow from the fact that two different low degree polynomials disagree on a large fraction of their domain.

Assume that  $E_0, E_1$  occur.

If for some  $j \in [r]$  we have  $\tilde{\rho} = f_j \circ \Gamma$ , then for that  $j$  we have  $a_i = \tilde{\rho}(t_i, 1, 0, 0) = f_j(\Gamma(t_i, 1, 0, 0)) = f_j(\gamma(t_i)) = f_j(x_i)$ . Hence in this case  $E_1$  doesn't occur. Hence, our assumption implies that  $\forall j \in [r] : \tilde{\rho} \neq f_j \circ \Gamma$ .

Since  $\tilde{\rho}$  and  $f_j \circ \Gamma$  are both polynomials of degree at most  $m \cdot h \cdot (q+1)$ , we know that  $\tilde{\rho}$  agrees with each  $f_j \circ \Gamma$  on a fraction of at most  $m \cdot h \cdot (q+1)/|\mathbb{F}|$  of the points. Hence,  $\tilde{\rho}$  agrees with  $f_j \circ \Gamma$  for some  $j$ , on a fraction of at most  $m \cdot h \cdot (q+1) \cdot r/|\mathbb{F}| \leq O(\delta)$  of the points. (Note that the size of the field  $\mathbb{F}$  was chosen to be  $h^c$  for a large enough  $c$ , and we can take  $c$  to be large enough so that the inequality holds). Call these points bad point.

Let  $t \in_R \mathbb{F} \setminus \{t_1, \dots, t_q\}$  and  $\alpha_1, \alpha_2, \alpha_3 \in_R \mathbb{F}$  and let  $z = \alpha_1 \gamma(t) + \alpha_2 z_2 + \alpha_3 z_3$  as in Step 7 of the protocol. Thus,  $\tilde{\pi}_1(z) = \tilde{\pi}_1(\alpha_1 \gamma(t) + \alpha_2 z_2 + \alpha_3 z_3)$ . If  $\tilde{\pi}_1(z)$  is different than  $\tilde{\rho}(t, \alpha_1, \alpha_2, \alpha_3)$  then the protocol rejects and  $E_0$  doesn't occur. Hence, our assumption implies that  $\tilde{\pi}_1(z) = \tilde{\rho}(t, \alpha_1, \alpha_2, \alpha_3)$ .

Assume that  $E_0, E_1$  occur and that  $(t, \alpha_1, \alpha_2, \alpha_3)$  is not a bad point. Then for every  $j$ , we have  $\tilde{\pi}_1(z) = \tilde{\rho}(t, \alpha_1, \alpha_2, \alpha_3) \neq f_j \circ \Gamma(t, \alpha_1, \alpha_2, \alpha_3) = f_j(z)$ , so the event  $E_2$  occurs. Thus, if  $E_0, E_1$  occur,  $\neg E_2$  can occur only if  $(t, \alpha_1, \alpha_2, \alpha_3)$  is a bad point, and recall that the fraction of bad points is at most  $O(\delta)$ . Hence,  $\neg E_2 \wedge E_1 \wedge E_0$  occurs with probability of at most  $O(\delta)$ . ■

**Claim 6.3.**

$$\Pr[\neg E_1 \wedge E_0] \leq O(\delta).$$

**Proof of Claim 6.3:** The proof will follow by the soundness property of the interactive PCP  $(P_0, V_0)$ .

For every  $j \in [r]$ , define  $\sigma_j : [p] \rightarrow \mathbb{F}$  by  $\sigma_j(Q) = f_j(E(Q))$ . We can view the pair  $\sigma_j, \tilde{P}_1$  as a (cheating) prover  $\tilde{P}_{0,j}$  for the interactive PCP  $(P_0, V_0)$  as follows. The prover  $\tilde{P}_{0,j}$  generates the string  $\sigma_j$  and interacts with the verifier  $V_0$  as  $\tilde{P}_1$  does in Step 2 of the protocol. After the interaction,  $V_0$  queries the string  $\sigma_j$  at the points  $Q_1, \dots, Q_q$ . We assume that if one of these points contains a value not in  $\{0, 1\}$  the verifier  $V_0$  rejects. Otherwise,  $V_0$  accepts or rejects according to his protocol. By the soundness property of the interactive PCP  $(P_0, V_0)$ , we know that none of these protocols accepts with probability larger than  $s$ .

Assume that  $\Pr[\neg E_1 \wedge E_0] > \alpha$ . Thus, with probability of at least  $\alpha$  there exists  $j \in [r]$ , such that  $(a_1, \dots, a_q) = (f_j(x_1), \dots, f_j(x_q))$  and the verifier  $V_0$  accepts on answers  $a_1, \dots, a_q$  given for the queries  $Q_1, \dots, Q_q$ , (after interacting with  $\tilde{P}_1$ ). (Where  $x_1, \dots, x_q$  are defined in Step 3 of the protocol and  $a_1, \dots, a_q$  are the answers obtained in Step 8 of the protocol).

Hence, there exists  $j_0$ , such that with probability of at least  $\alpha/r$  we have that  $(a_1, \dots, a_q) = (f_{j_0}(x_1), \dots, f_{j_0}(x_q)) = (\sigma_{j_0}(Q_1), \dots, \sigma_{j_0}(Q_q))$  and the verifier  $V_0$  accepts on answers  $a_1, \dots, a_q$  given for the queries  $Q_1, \dots, Q_q$ , (after interacting with  $\tilde{P}_1$ ). Thus, the verifier  $V_0$  accepts with probability at least  $\alpha/r$  after interacting with  $\tilde{P}_{0,j_0}$ . Hence,  $\alpha/r \leq s$ , or equivalently  $\alpha \leq rs \leq 2\sqrt{s} = O(\delta)$ . ■

We can now conclude that

$$\begin{aligned} \Pr[E_0] &= \Pr[E_0 \wedge \neg E_1] + \Pr[E_0 \wedge E_1] = \\ &= \Pr[E_0 \wedge \neg E_1] + \Pr[E_0 \wedge E_1 \wedge \neg E_2] + \Pr[E_0 \wedge E_1 \wedge E_2] \leq \\ &= \Pr[E_0 \wedge \neg E_1] + \Pr[E_0 \wedge E_1 \wedge \neg E_2] + \Pr[E_0 \wedge E_2] \leq O(\delta). \end{aligned}$$

This proves the soundness property of the protocol  $(P_1, V_1)$ .

## The final protocol

The protocol  $(P_1, V_1)$  has the required properties, except that the one query made by  $V_1$  is to  $\pi_1$  and hence the query returns a field element in  $\mathbb{F}$ , rather than a single bit. We will convert  $(P_1, V_1)$  into the final protocol  $(P, V)$ . For the final protocol, we will use the Hadamard error correcting code  $Had : \mathbb{F} \rightarrow \{0, 1\}^{|\mathbb{F}|}$ . (We could use any other error correcting code with good list-decoding properties).

Formally, given  $y, z \in \mathbb{F}$ , we think of  $y, z$  as strings of length  $\log |\mathbb{F}|$  bits (e.g., by thinking of  $\mathbb{F}$  as a vector space over  $\mathbb{GF}[2]$ ). The function  $Had : \mathbb{F} \rightarrow \{0, 1\}^{|\mathbb{F}|}$  is defined by  $Had(y)_z = (y, z)$ , where  $(y, z)$  denotes the scalar product of  $y$  and  $z$  over  $\mathbb{GF}[2]$ .

For every  $\eta \in \{0, 1\}^{|\mathbb{F}|}$  and every  $\delta \geq 0$ , denote by  $N_\delta[\eta] \subseteq \mathbb{F}$ , the set of all elements  $y \in \mathbb{F}$  such that  $Had(y)$  is  $1/2 - \delta$  close to  $\eta$ . That is,

$$N_\delta[\eta] = \left\{ y \in \mathbb{F} : \Pr_{z \in \mathbb{F}}[Had(y)_z = \eta_z] \geq 1/2 + \delta \right\}.$$

It is well known, by Parseval equality, that for every  $\eta, \delta$ ,

$$|N_\delta[\eta]| < \frac{1}{\delta^2}.$$

Let  $\pi$  be the concatenation of  $Had(y)$  for all the elements  $y$  in the truth table of  $\pi_1$ . That is,

$$\pi = (Had(\pi_1(1)), \dots, Had(\pi_1(p_1))).$$

In the final protocol, the prover  $P$  generates  $\pi$ , rather than  $\pi_1$ . The protocol  $(P, V)$  is then the same as  $(P_1, V_1)$ , except that the verifier  $V$ , rather than querying  $\pi_1$  at a point  $Q$ , asks the prover  $P$  to send the value of  $\pi_1(Q)$ , and verifies the answer by reading a single random bit in  $Had(\pi_1(Q))$  (i.e., a single random bit of  $\pi(Q)$ ). Formally, if  $P$  answers by  $y$ , the verifier  $V$  compares  $Had(y)_z$  with  $\pi(Q)_z$ , for a random  $z \in \mathbb{F}$ , and rejects if they are different. If they are the same,  $V$  continues as  $V_1$ , using  $y$  instead of  $\pi_1(Q)$ , and accepts iff  $V_1$  accepts.

Note that the size of  $\pi$  is at most the size of  $\pi_1$  times  $\text{poly}(h) \leq \text{poly}(p)$ . Note that the prover only needs to communicate one additional field element ( $O(\log h)$  bits). Note also that the completeness of the algorithm remains the same. Thus, it remains to prove the soundness property of the final protocol.

The soundness property of the final protocol will follow from the list decoding properties of the Hadamard code, and from the soundness property of the protocol  $(P_1, V_1)$ . Assume that the instance  $x$  is not in  $L$ . Let  $\tilde{P}$  be any (cheating) prover and let  $\tilde{\pi} : [p_1] \times |\mathbb{F}| \rightarrow \{0, 1\}$  be any string. We will use  $\tilde{P}, \tilde{\pi}$  to define a (cheating) prover  $\tilde{P}_1$ , and a string  $\tilde{\pi}_1$  for the protocol  $(P_1, V_1)$ .

Fix

$$\delta = \sqrt[3]{s_1}$$

(where  $s_1$  is the soundness parameter of  $(P_1, V_1)$ ). We define  $\tilde{\pi}_1$  to be a (probabilistic) string as follows. For every  $Q \in [p_1]$ , define  $\tilde{\pi}(Q) \in \{0, 1\}^{|\mathbb{F}|}$  by  $\tilde{\pi}(Q)_z = \tilde{\pi}(Q, z)$ . Define  $\tilde{\pi}_1(Q)$  to be a random element of  $N_\delta[\tilde{\pi}(Q)]$  if  $N_\delta[\tilde{\pi}(Q)]$  is not empty, and an arbitrary value otherwise. The prover  $\tilde{P}_1$  acts the same as  $\tilde{P}$ , (except that  $\tilde{P}$  is asked by  $V$  for the value of  $\pi_1(Q)$ , and  $\tilde{P}_1$  doesn't have to supply this answer because  $V_1$  queries  $\pi_1$  for this value directly).

Denote by  $A$  the event that  $(\tilde{P}(x), V^{\tilde{\pi}}(x))$  accepts (or, for simplicity,  $V$  accepts), and denote

$$\alpha = \Pr[A] = \Pr[(\tilde{P}(x), V^{\tilde{\pi}}(x)) = 1],$$

where the probability is over the coin tosses.

Denote by  $A_1$  the event that  $(\tilde{P}_1(x), V_1^{\tilde{\pi}_1}(x))$  accepts (or, for simplicity,  $V_1$  accepts), and denote

$$\alpha_1 = \Pr[A_1] = \Pr[(\tilde{P}_1(x), V_1^{\tilde{\pi}_1}(x)) = 1],$$

where the probability is over the coin tosses, and over the choice of the string  $\tilde{\pi}_1$  (recall that  $\tilde{\pi}_1$  was defined probabilistically). By the soundness property of the protocol  $(P_1, V_1)$ , we know that

$$\alpha_1 \leq s_1.$$

We will bound  $\alpha$  as a function of  $\alpha_1, \delta$ . This is done as follows.

By the definition of  $(P, V)$ , the verifier  $V$  asks the prover  $\tilde{P}$  for the value of  $\pi_1(Q)$  and then verifies that the answer is consistent with  $\tilde{\pi}(Q)_z$ , for a random  $z \in \mathbb{F}$ . Denote by  $y$  the answer given by  $\tilde{P}$  (for the query  $Q$ ). Denote the following events:

1. Event  $B$ :  $y \notin N_\delta[\tilde{\pi}(Q)]$
2. Event  $C$ :  $\neg B$  occurs (i.e.,  $y \in N_\delta[\tilde{\pi}(Q)]$ ) and  $\tilde{\pi}_1(Q) = y$ .

Note that:

1.  $\Pr[A|B] \leq 1/2 + \delta$   
(If  $B$  occurs then  $Had(y)$  is not  $1/2 - \delta$  close to  $\tilde{\pi}(Q)$ . Hence  $V$  accepts with probability of at most  $1/2 + \delta$ , since he compares  $Had(y)_z$  with  $\tilde{\pi}(Q)_z$  for a random  $z$ ).
2.  $\Pr[C|\neg B] > \delta^2$   
(If  $\neg B$  occurs then  $y \in N_\delta[\tilde{\pi}(Q)]$ . Since  $\tilde{\pi}_1(Q)$  is a random element of  $N_\delta[\tilde{\pi}(Q)]$ , with probability of  $1/|N_\delta[\tilde{\pi}(Q)]| > \delta^2$ , we have  $\tilde{\pi}_1(Q) = y$ ).
3.  $\Pr[A|C] \leq \Pr[A_1|C]$   
(If  $C$  occurs then  $y \in N_\delta[\tilde{\pi}(Q)]$  and  $\tilde{\pi}_1(Q) = y$ . In this case by the definition of  $V$ , if  $V_1$  rejects on the oracle answer  $\tilde{\pi}_1(Q)$  then  $V$  rejects on the prover answer  $y$ ).
4.  $\Pr[A|\neg B] = \Pr[A|C]$   
(If  $\neg B$  occurs then  $y \in N_\delta[\tilde{\pi}(Q)]$ . The value of  $\tilde{\pi}_1(Q)$  was chosen at random from  $N_\delta[\tilde{\pi}(Q)]$ , independently of any other coin tosses. The event  $A$  doesn't depend on the string  $\tilde{\pi}_1$  at all (and hence not on the event  $\tilde{\pi}_1(Q) = y$ )).

By the above, we can bound

$$\begin{aligned} \alpha &= \Pr[A] \leq \Pr[A|B] + \Pr[A|\neg B] \cdot \Pr[\neg B] \\ &= \Pr[A|B] + \Pr[A|C] \cdot \Pr[\neg B] \\ &\leq \Pr[A|B] + \Pr[A_1|C] \cdot \Pr[\neg B] \end{aligned}$$

Since  $C \subseteq \neg B$ , and by the above, we can also bound

$$\begin{aligned} \alpha_1 &= \Pr[A_1] \geq \Pr[A_1|C] \cdot \Pr[C] \\ &= \Pr[A_1|C] \cdot \Pr[C|\neg B] \cdot \Pr[\neg B] \\ &\geq \Pr[A_1|C] \cdot \delta^2 \cdot \Pr[\neg B] \end{aligned}$$

Hence,

$$\begin{aligned}\alpha &\leq \Pr[A|B] + \Pr[A_1|C] \cdot \Pr[\neg B] \\ &\leq 1/2 + \delta + \alpha_1/\delta^2 \leq 1/2 + \delta + s_1/\delta^2.\end{aligned}$$

Since  $\delta = \sqrt[3]{s_1}$ , we get

$$\alpha \leq 1/2 + O(\sqrt[3]{s_1}) = 1/2 + O(\sqrt[6]{s}) + O(\sqrt[3]{\epsilon}).$$

This proves the soundness property of Theorem 4 because we could start from  $\epsilon^3$  rather than  $\epsilon$ , without changing the guarantees in the statement of the theorem.

### The moreover part

As for the moreover part of Theorem 4, the first one is immediate from the definitions of  $\pi_0$  and  $\pi$ . For the second one, note that  $V$  can operate as follows. In the first phase,  $V$  runs the first phase of the verifier  $V_0$  and generates the same string  $g$  that is generated by  $V_0$ . In the second phase,  $V$  operates according to the definitions of the protocols  $(P_1, V_1)$  and  $(P, V)$ , and note that this can be done in time  $\text{poly}(T_2, |\mathbb{F}|)$ , as it only requires running the second phase of  $V_0$  and polynomial time operations on objects of size  $\text{poly}(|\mathbb{F}|)$ . ■

### Interactive PCP for Arithmetic Formulas

The following theorem, which is one of our main results, follows immediately by Theorem 2 and Theorem 4.

**Theorem 5.** *For any<sup>11</sup>  $\epsilon \geq 1/n$ ,*

$$\text{SAT}_{N,k,d} \in \text{IPCP}(p, q, \ell, c, s),$$

*with  $p = \text{poly}(k, d, 1/\epsilon)$ ,  $q = 1$ ,  $\ell = \text{poly}(\log k, d, \log N, 1/\epsilon)$ ,  $c = 1$ , and  $s = 1/2 + \epsilon$ .*

*Moreover, the following two properties can be attained.*

1. *The string  $\pi$  (generated by the prover in the first round of the protocol) depends only on the witness  $w = (w_1, \dots, w_k)$  and on the parameters  $d, \epsilon$ , and not on the instance  $\Psi_1, \dots, \Psi_N$ .*
2. *The verifier can partition his work into two phases: In the first phase he runs in time  $\leq \text{poly}(|\Psi_1| + \dots + |\Psi_N|)$ , and generates a string  $h$  of size  $\leq \text{poly}(\log N, \log k, d, \log(1/\epsilon))$ . This phase depends only on  $\Psi_1, \dots, \Psi_N$  and on the parameters (and on the randomness of the verifier) and does not depend on the oracle  $\pi$  or on the interaction with the prover. In the second phase, which is the interactive phase, the verifier uses only the string  $h$  generated in the first phase (and not the instance  $\Psi_1, \dots, \Psi_N$ ), and runs in time  $\leq \text{poly}(\log N, \log k, d, 1/\epsilon)$ . The messages that he sends in this phase, and the oracle queries, depend only on  $h$  and are independent of the messages sent by the prover or the oracle answers.*

We can now use Theorem 5 to farther improve Theorem 2.

---

<sup>11</sup>We require  $\epsilon \geq 1/n$  in order to ensure that the prover and verifier run in time polynomial in the size of the instance. We could take  $1/n > \epsilon > 0$  and then the running time is polynomial in  $1/\epsilon$

**Theorem 6.** For any soundness parameter<sup>12</sup>  $s \geq 2^{-n}$ ,

$$\text{SAT}_{N,k,d} \in \text{IPCP}(p, q, \ell, c, s),$$

with  $p = \text{poly}(k, d)$ ,  $q = \text{poly}(\log \frac{1}{s})$ ,  $\ell = \text{poly}(\log k, d, \log N, \log \frac{1}{s})$ , and  $c = 1$ .

Moreover, the following two properties can be attained.<sup>13</sup>

1. The string  $\pi$  (generated by the prover in the first round of the protocol) depends only on the witness  $w = (w_1, \dots, w_k)$  and on the parameter  $d$ , and not on the instance  $\Psi_1, \dots, \Psi_N$ .
2. The verifier can partition his work into two phases: In the first phase he runs in time  $\leq \text{poly}(|\Psi_1| + \dots + |\Psi_N|)$ , and generates a string  $h$  of size  $\leq \text{poly}(\log N, \log k, d, \log \frac{1}{s})$ . This phase depends only on  $\Psi_1, \dots, \Psi_N$  and on the parameters (and on the randomness of the verifier) and does not depend on the oracle  $\pi$  or on the interaction with the prover. In the second phase, which is the interactive phase, the verifier uses only the string  $h$  generated in the first phase (and not the instance  $\Psi_1, \dots, \Psi_N$ ), and runs in time  $\leq \text{poly}(\log N, \log k, d, \log \frac{1}{s})$ . The messages that he sends in this phase, and the oracle queries, depend only on  $h$  and are independent of the messages sent by the prover or the oracle answers.

**Proof of Theorem 6:** Theorem 6 follows as a corollary of Theorem 5 (with, say,  $\epsilon = 1/4$ ), by repeating the protocol sequentially  $\text{poly}(\log \frac{1}{s})$  times.

■

## 7 Interactive PCP for Constant Depth Formulas

In this section we prove the following two theorems.

**Theorem 7.** Let  $L = \{x : \exists w \text{ s.t. } \mathcal{R}_L(x, w) = 1\}$  be an NP language, such that  $\mathcal{R}_L$  is a polynomial size constant depth Boolean formula, over the gates  $\{\neg, \vee, \wedge, \oplus\}$ . Then, for any<sup>14</sup>  $\epsilon \geq 1/n$  and any  $\delta \geq 2^{-n}$ ,

$$L \in \text{IPCP}(p, q, \ell, c, s),$$

with  $p = \text{poly}(k, \frac{1}{\epsilon}, \log \frac{1}{\delta})$ ,  $q = 1$ ,  $\ell = \text{poly}(\log n, \frac{1}{\epsilon}, \log \frac{1}{\delta})$ ,  $c \geq 1 - \delta$ , and  $s \leq \frac{1}{2} + \epsilon$  (where  $n$  is the instance size and  $k$  is the witness size).

Moreover, the following two properties can be attained.

1. The string  $\pi$  (generated by the prover in the first round of the protocol) depends only on the witness  $w = (w_1, \dots, w_k)$  and on the parameters  $n, \epsilon, \delta$ , and not on the instance  $x$ .
2. The verifier can partition his work into two phases: In the first phase he runs in time  $\leq \text{poly}(n)$ , and generates a string  $h$  of size  $\leq \text{poly}(\log n, \log \frac{1}{\epsilon}, \log \frac{1}{\delta})$ . This phase depends only on  $x$  and on the parameters (and on the randomness of the verifier), and does not depend on

<sup>12</sup>We require  $s \geq 2^{-n}$  in order to ensure that the prover and verifier run in time polynomial in the size of the instance. We could take  $2^{-n} > s > 0$  and then the running time is polynomial in  $\log \frac{1}{s}$ .

<sup>13</sup>The second property is not needed for the main result. It is an additional feature that may be important for applications. We use it for the application of zero-knowledge.

<sup>14</sup>We require  $\epsilon \geq 1/n$  and  $\delta \geq 2^{-n}$  in order to ensure that the prover and verifier run in time polynomial in the size of the instance. We could take  $1/n > \epsilon > 0$  and  $2^{-n} > \delta > 0$  and then the running time is polynomial in  $1/\epsilon$  and  $\log \frac{1}{\delta}$ .



the oracle  $\pi$  or on the interaction with the prover. In the second phase, which is the interactive phase, the verifier uses only the string  $h$  generated in the first phase (and not the instance  $x$ ), and runs in time  $\leq \text{poly}(\log n, \frac{1}{\epsilon}, \log \frac{1}{\delta})$ . The messages that he sends in this phase, and the oracle queries, depend only on  $h$  and are independent of the messages sent by the prover or the oracle answers.

**Theorem 8.** *Let  $L = \{x : \exists w \text{ s.t. } \mathcal{R}_L(x, w) = 1\}$  be an NP language, such that  $\mathcal{R}_L$  is a polynomial size constant depth Boolean formula, over the gates  $\{\neg, \vee, \wedge, \oplus\}$ . Then, for any<sup>15</sup>  $\delta, \epsilon \geq 2^{-n}$ ,*

$$L \in \text{IPCP}(p, q, \ell, c, s),$$

with  $p = \text{poly}(k, \log \frac{1}{\delta})$ ,  $q = \text{poly}(\log \frac{1}{\epsilon})$ ,  $\ell = \text{poly}(\log n, \log \frac{1}{\delta}, \log \frac{1}{\epsilon})$ ,  $s \leq \epsilon + \delta$ , and  $c \geq 1 - \delta$  (where  $n$  is the instance size and  $k$  is the witness size).

Moreover, the following two properties can be attained.

1. The string  $\pi$  (generated by the prover in the first round of the protocol) depends only on the witness  $w = (w_1, \dots, w_k)$  and on the parameters  $n, \delta$ , and not on the instance  $x$ .
2. The verifier can partition his work into two phases: In the first phase he runs in time  $\leq \text{poly}(n)$ , and generates a string  $h$  of size  $\leq \text{poly}(\log n, \log \frac{1}{\delta}, \log \frac{1}{\epsilon})$ . This phase depends only on  $x$  and on the parameters (and on the randomness of the verifier), and does not depend on the oracle  $\pi$  or on the interaction with the prover. In the second phase, which is the interactive phase, the verifier uses only the string  $h$  generated in the first phase (and not the instance  $x$ ), and runs in time  $\leq \text{poly}(\log n, \log \frac{1}{\delta}, \log \frac{1}{\epsilon})$ . The messages that he sends in this phase, and the oracle queries, depend only on  $h$  and are independent of the messages sent by the prover or the oracle answers.

**Remark:** Note that Theorems 7 and 8 imply, in particular, that there exists an interactive PCP for proving the satisfiability of a constant depth Boolean formula (over the gates  $\{\neg, \vee, \wedge, \oplus\}$ ), with parameters  $(p, q, \ell, c, s)$ , as stated in the above two theorems.

For the proof of Theorems 7 and 8, we will approximate a constant depth Boolean formula by an arithmetic formula of low degree. This is a well known method in complexity theory, originated by [R87, S87] (see [B93] for a survey). It is well known that a constant depth formula can be approximated by a low degree polynomial, using a small number of random bits. Although this is usually not mentioned explicitly, it is easy to check that the approximating low degree polynomial can actually be computed by a polynomial size arithmetic formula.

**Lemma 7.1.** *There exists a probabilistic Turing machine  $\mathcal{M}$ , that takes as input a constant depth Boolean formula  $C : \{0, 1\}^k \rightarrow \{0, 1\}$  of size  $m$  (over the gates  $\vee, \wedge, \neg, \oplus$ ), and a parameter  $\delta > 0$ . It runs in time  $\text{poly}(m, \log \frac{1}{\delta})$ , and uses  $\text{poly}(\log m, \log \frac{1}{\delta})$  random bits. It outputs an arithmetic formula of size<sup>16</sup>  $\text{poly}(m, \log \frac{1}{\delta})$  and of degree  $\text{poly}(\log m, \log \frac{1}{\delta})$ , such that for every  $C, \delta$  and every  $x \in \{0, 1\}^k$ ,*

$$\Pr[\mathcal{M}(C, \delta)(x) = C(x)] \geq 1 - \delta.$$

<sup>15</sup>We require  $\delta, \epsilon \geq 2^{-n}$  in order to ensure that the prover and verifier run in time polynomial in the size of the instance. We could take  $2^{-n} > \delta, \epsilon > 0$  and then the running time is polynomial in  $\log \frac{1}{\delta}$  and  $\log \frac{1}{\epsilon}$ .

<sup>16</sup>The size and the degree of the arithmetic formula (and hence also the running time of  $\mathcal{M}$ ) depend exponentially on the depth of  $C$  (which is assumed to be constant).

**Proof of Lemma 7.1:** Fix a constant depth Boolean formula  $C : \{0, 1\}^k \rightarrow \{0, 1\}$  of size  $m$  (over the gates  $\vee, \wedge, \neg, \oplus$ ), and fix a parameter  $\delta > 0$ . The Turing machine  $\mathcal{M}$ , on input  $(C, \delta)$ , operates as follows:

1. Convert  $C$  into a Boolean formula  $C_1 : \{0, 1\}^k \rightarrow \{0, 1\}$  over the gates  $\vee, \neg, \oplus$ , by using De Morgan's law. The resulting Boolean formula  $C_1$  is of constant depth, is of size  $O(m)$ , and computes exactly the same function as  $C$ .
2. Convert  $C_1$  into a Boolean formula  $C_2 : \{0, 1\}^k \rightarrow \{0, 1\}$  such that the fan-in of each  $\vee$  gate is exactly  $m$ . This is done by adding at most  $\text{poly}(m)$  dummy gates that consist of the constant 0. The resulting Boolean formula  $C_2$  is of constant depth, is of size  $\leq \text{poly}(m)$ , and computes exactly the same function as  $C_1$ .
3. Approximate  $C_2$  by an arithmetic circuit  $\Psi : \{0, 1\}^k \rightarrow \{0, 1\}$  (over  $\mathbb{GF}[2]$ ), by replacing each Boolean gate by an arithmetic gate, as follows:

- (a) Replace each negation gate  $\neg x$  (where  $x$  is the input to the gate) by  $1 - x$ . This does not change the functionality of  $C_2$ .
- (b) Replace each XOR gate  $\oplus_{i=1}^t x_i$  (where  $x_1, \dots, x_t$  are the inputs to the gate) by  $\sum_{i=1}^t x_i$ . This does not change the functionality of  $C_2$  (since the sum is over  $\mathbb{GF}[2]$ ).
- (c) Let  $ECC : \{0, 1\}^m \rightarrow \{0, 1\}^{100m}$  be a linear error correcting code with relative distance  $1/3$ . Note that the linearity of  $ECC$  implies that it can be computed by a constant depth arithmetic circuit of degree 1 and size  $\text{poly}(m)$  (over  $\mathbb{GF}[2]$ ).

Denote by  $m'$  the number of  $\vee$  gates in  $C_2$ . (Note that  $m' \leq m$ ). Fix  $l = O(\log m' + \log \frac{1}{\delta})$  such that

$$\left(\frac{2}{3}\right)^l \leq \frac{\delta}{m'}.$$

Choose independently at random  $i_1, \dots, i_l \in_R [100m]$  (each  $i_j$  corresponds to a random coordinate of a codeword). Replace each  $\bigvee_{i=1}^m x_i$  by

$$\eta(x_1, \dots, x_m) \stackrel{\text{def}}{=} 1 - \prod_{j=1}^l (1 - ECC(x_1, \dots, x_m)_{i_j}).$$

Notice that we approximate each  $\vee$  gate using the same random bits. Also notice that  $\eta$  can be computed by a constant depth arithmetic circuit of size  $\text{poly}(m, l) = \text{poly}(m, \log \frac{1}{\delta})$ , and that for every  $x = (x_1, \dots, x_m) \in \{0, 1\}^m$ ,

$$\Pr \left[ \eta(x_1, \dots, x_m) \neq \bigvee_{i=1}^m x_i \right] \leq \left(\frac{2}{3}\right)^l \quad (12)$$

(where the probability is over the random choices of  $i_1, \dots, i_l \in_R [100m]$ ).

This arithmetization procedure results with a constant depth arithmetic circuit  $\Psi : \{0, 1\}^k \rightarrow \{0, 1\}$  of size  $\text{poly}(m, l) \leq \text{poly}(m, \log \frac{1}{\delta})$ , and of degree  $\leq l^{O(1)} \leq \text{poly}(\log m, \log \frac{1}{\delta})$ . We stress that  $\Psi$  is an arithmetic *circuit* (as opposed to a formula), since each  $\vee$  gate was replaced by a constant depth circuit. The fact that  $\Psi$  is of constant depth implies that it can be converted (in polynomial time) into an arithmetic *formula* of the same degree, and with only a polynomial increase in its size.

We next prove that for every  $x = (x_1, \dots, x_k) \in \{0, 1\}^k$ ,

$$\Pr[\Psi(x_1, \dots, x_k) = C(x_1, \dots, x_k)] \geq 1 - \delta$$

(where the probability is over the random choices of  $i_1, \dots, i_l \in_R [100m]$ ).

Fix any input  $x = (x_1, \dots, x_k) \in \{0, 1\}^k$ . Let  $g_1, \dots, g_{m'}$  be all the  $\vee$  gates in  $C_2$  ordered so that for every  $i, j \in [m']$  if the gate  $g_i$  is a descendant of  $g_j$  then  $i < j$ . Let  $\eta_1, \dots, \eta_{m'}$  be their corresponding approximations in  $\Psi$ . For  $i = 1, \dots, m'$ , let  $E_i$  denote the event that the output of  $g_i$  when evaluating  $C_2$  on input  $x$  is the same as the output of  $\eta_i$  when evaluating  $\Psi$  on input  $x$ . Then,

$$\begin{aligned} & \Pr[\Psi(x_1, \dots, x_k) = C(x_1, \dots, x_k)] \geq \\ & \Pr[E_1 \wedge E_2 \wedge \dots \wedge E_{m'}] = \\ & \Pr[E_1] \cdot \Pr[E_2|E_1] \cdot \dots \cdot \Pr[E_{m'}|E_1 \wedge \dots \wedge E_{m'-1}] \geq \\ & \prod_{i=1}^{m'} \left(1 - \left(\frac{2}{3}\right)^l\right) \geq \\ & \prod_{i=1}^{m'} \left(1 - \frac{\delta}{m'}\right) \geq \\ & 1 - \delta. \end{aligned}$$

It remains to note that  $\mathcal{M}$  runs in time  $\leq \text{poly}(m, \log \frac{1}{\delta})$ , and uses  $\leq \text{poly}(\log m, \log \frac{1}{\delta})$  random bits. ■

We next use Lemma 7.1 to prove Theorem 7 and Theorem 8.

**Proof of Theorem 7:** Fix parameters  $\epsilon \geq 1/n$  and  $\delta \geq 2^{-n}$ . Say the prover wishes to prove that  $x \in L$ , for some string  $x \in \{0, 1\}^n$ . Let  $k$  denote the size of the witness. Thus,  $R_L(x, \cdot)$  is a function

$$R_L(x, \cdot) : \{0, 1\}^k \rightarrow \{0, 1\}.$$

Assume that  $k \geq \log n$ . This is without loss of generality since otherwise, determining whether  $x \in L$  can be done in polynomial time. Theorem 7 follows from Theorem 5 and Lemma 7.1, as follows.

Let  $\delta' \stackrel{\text{def}}{=} \min(\delta, \frac{\epsilon}{2})$  and let  $\epsilon' \stackrel{\text{def}}{=} \frac{\epsilon}{2}$ . Let  $d$  be the degree of the arithmetic formula obtained by applying Lemma 7.1 with the Boolean formula  $R_L(x, \cdot)$  and the parameter  $\delta'$ . Note that

$$d = \text{poly}\left(\log n, \log \frac{1}{\delta'}\right) = \text{poly}\left(\log n, \log \frac{1}{\delta}, \log \frac{1}{\epsilon}\right).$$

The prover generates  $\pi$  as in Theorem 5, with parameters  $k, d, \epsilon'$ . Thus,

$$|\pi| \leq \text{poly}\left(k, \frac{1}{\epsilon}, \log \frac{1}{\delta}\right).$$

In the interactive phase, the verifier runs the (probabilistic) Turing machine  $\mathcal{M}$  (from Lemma 7.1) on input  $R_L(x, \cdot)$  and  $\delta'$ . Namely, the verifier uses  $\text{poly}(\log n, \log \frac{1}{\delta'}) = \text{poly}(\log n, \log \frac{1}{\delta}, \log \frac{1}{\epsilon})$  random bits and generates an arithmetic formula

$$\Psi : \{0, 1\}^k \rightarrow \{0, 1\}$$

of size  $\text{poly}(n, \log \frac{1}{\delta'}) = \text{poly}(n, \log \frac{1}{\delta}, \log \frac{1}{\epsilon})$  and degree  $d$ . Recall that for any given input (and in particular for the input encoded in  $\pi$ ),  $\Psi$  and  $R_L(x, \cdot)$  agree with probability at least  $1 - \delta'$ . The verifier sends these random bits to the prover. The prover can use these random bits to compute  $\Psi$  on his own. Next, the prover and verifier run the interactive protocol of Theorem 5 with respect to  $\Psi$  and the parameter  $\epsilon'$  (and with  $N = 1$ ).

The fact that this protocol guarantees the parameters  $(p, q, \ell, c, s)$ , as stated in the theorem, follows from Theorem 5 with the parameter  $\epsilon'$ , and from Lemma 7.1 with the parameter  $\delta'$ . In particular, note that we get

$$c \geq 1 - \delta' \geq 1 - \delta,$$

and

$$s \leq \frac{1}{2} + \epsilon' + \delta' \leq \frac{1}{2} + \epsilon.$$

As for the moreover part of Theorem 7, the first one follows directly from Theorem 5. For the second one, the verifier can operate as follows. In the first phase, he chooses the random bits used to generate  $\Psi$ , and emulates the first phase verifier of Theorem 5 with respect to  $\Psi$  and  $\epsilon'$ . The string  $h$  consists of these random bits, together with the corresponding string generated in the first phase of Theorem 5 (with respect to  $\Psi$  and  $\epsilon'$ ). In the second phase, the verifier sends to the prover the random bits used to generate  $\Psi$ , and emulates the second phase verifier of Theorem 5 (with respect to  $\Psi$  and  $\epsilon'$ ). The parameters in the moreover part of Theorem 7 follow from the parameters of Theorem 5, together with the fact that the number of random bits used to generate  $\Psi$  is at most  $\text{poly}(\log n, \log \frac{1}{\epsilon}, \log \frac{1}{\delta})$ . ■

We next prove Theorem 8. We note that the proof is very similar to the proof of Theorem 7.

**Proof of Theorem 8:** Fix parameters  $\delta, \epsilon \geq 2^{-n}$ . Say the prover wishes to prove that  $x \in L$ , for some string  $x \in \{0, 1\}^n$ . Let  $k$  denote the size of the witness. Thus,  $R_L(x, \cdot)$  is a function

$$R_L(x, \cdot) : \{0, 1\}^k \rightarrow \{0, 1\}.$$

Assume that  $k \geq \log n$ . This is without loss of generality since otherwise, determining whether  $x \in L$  can be done in polynomial time. Theorem 8 follows from Theorem 6 and Lemma 7.1, as follows.

Let  $d$  be the degree of the arithmetic formula obtained by applying Lemma 7.1 with the Boolean formula  $R_L(x, \cdot)$  and the parameter  $\delta$ . Thus,

$$d = \text{poly} \left( \log n, \log \frac{1}{\delta} \right).$$

The prover generates  $\pi$  as in Theorem 6, with parameters  $k, d$ . Thus,

$$|\pi| \leq \text{poly} \left( k, \log \frac{1}{\delta} \right).$$

In the interactive phase, the verifier runs the (probabilistic) Turing machine  $\mathcal{M}$  (from Lemma 7.1) on input  $R_L(x, \cdot)$  and  $\delta$ . Namely, the verifier uses  $\text{poly}(\log n, \log \frac{1}{\delta})$  random bits and generates an arithmetic formula

$$\Psi : \{0, 1\}^k \rightarrow \{0, 1\}$$

of size  $\text{poly}(n, \log \frac{1}{\delta})$  and degree  $d$ . Recall that for any given input (and in particular for the input encoded in  $\pi$ ),  $\Psi$  and  $R_L(x, \cdot)$  agree with probability at least  $1 - \delta$ . The verifier sends these random

bits to the prover. The prover can use these random bits to compute  $\Psi$  on his own. Next, the prover and verifier run the interactive protocol of Theorem 6 with respect to  $\Psi$  and the parameter  $s = \epsilon$  (and with  $N = 1$ ).

The fact that this protocol guarantees the parameters  $(p, q, \ell, c, s)$ , as stated in Theorem 8, follows from Theorem 5 with the parameter  $s = \epsilon$ , and from Lemma 7.1 with the parameter  $\delta$ . In particular, note that we get

$$c \geq 1 - \delta,$$

and

$$s \leq \epsilon + \delta.$$

As for the moreover part of Theorem 8, the first one follows directly from Theorem 6. For the second one, the verifier can operate as follows. In the first phase, he chooses the random bits used to generate  $\Psi$ , and emulates the first phase verifier of Theorem 6 with respect to  $\Psi$  and  $s = \epsilon$ . The string  $h$  consists of these random bits, together with the corresponding string generated in the first phase of Theorem 6 (with respect to  $\Psi$  and  $s = \epsilon$ ). In the second phase, the verifier sends to the prover the random bits used to generate  $\Psi$ , and emulates the second phase verifier of Theorem 6 (with respect to  $\Psi$  and  $s = \epsilon$ ). The parameters in the moreover part of Theorem 8 follow from the parameters of Theorem 6, together with the fact that the number of random bits used to generate  $\Psi$  is at most  $\text{poly}(\log n, \log \frac{1}{\delta})$ . ■

## 8 Succinct Zero-Knowledge Proofs

In this section, we give an application of our results. We show that for some NP languages, with short witnesses, there exist succinct zero-knowledge proofs.

**Theorem 9.** *Fix a parameter  $t = t(n) \leq n$ , and assume the existence of a (one-way) function  $f : \{0, 1\}^t \rightarrow \{0, 1\}^t$  that can be computed in time  $\text{poly}(n)$  and cannot be inverted by  $\text{poly}(n)$ -time adversaries (except with  $\text{negl}(n)$  probability). Let  $L = \{x : \exists w \text{ s.t. } \mathcal{R}_L(x, w) = 1\}$  be an NP language, such that  $\mathcal{R}_L$  is a constant depth Boolean formula (over the gates  $\{\neg, \vee, \wedge, \oplus\}$ ). Then, the language  $L$  has an interactive zero-knowledge proof of size  $\text{poly}(t, k)$  (where  $k$  is the witness size), with soundness  $s \leq 2^{-t}$  and completeness  $c \geq 1 - 2^{-t}$ . In addition, the prover runs in polynomial time (assuming that he has a witness as an additional input).*

*Moreover, this zero-knowledge proof can be partitioned into two phases. The first phase is non-interactive. In this phase the prover sends to the verifier a certain (non-interactive) commitment to his witness at hand  $w$ . This message is of size  $\text{poly}(t, k)$  (where  $k = |w|$ ), and depends only on the witness  $w$ , and not on the instance  $x$ . The second phase is interactive. In this phase the prover and verifier engage in a zero-knowledge proof that indeed the string committed to is a valid witness for  $x$  (i.e.,  $\mathcal{R}_L(x, w) = 1$ ). This phase consists of  $\text{poly}(t, \log n)$  bits of interaction.*

Instead of proving Theorem 9 directly, we prove the following more general theorem.

**Theorem 10.** *Fix a parameter  $t = t(n) \leq n$ , and assume the existence of a (one-way) function  $f : \{0, 1\}^t \rightarrow \{0, 1\}^t$  that can be computed in time  $\text{poly}(n)$  and cannot be inverted by  $\text{poly}(n)$ -time adversaries (except with  $\text{negl}(n)$  probability). Then, the language  $\text{SAT}_{N, k, d}$  (defined in Section 5) has an interactive zero-knowledge proof of size  $\text{poly}(t, \log N, k, d)$ , with soundness  $s \leq 2^{-t}$  and completeness  $c = 1$ . In addition, the prover runs in polynomial time (assuming that he has a witness as an additional input).*

Moreover, this zero-knowledge proof can be partitioned into two phases. The first phase is non-interactive. In this phase the prover sends to the verifier a certain (non-interactive) commitment to his witness at hand  $w$ . This message is of size  $\text{poly}(t, k, d)$  (where  $k = |w|$ ), and depends only on the witness  $w$ , and not on the instance  $\Psi_1, \dots, \Psi_N$ . The second phase is interactive. In this phase the prover and verifier engage in a zero-knowledge proof that indeed the string committed to zeros all the arithmetic formulas  $\Psi_1, \dots, \Psi_N$  (i.e.,  $\bigwedge_{i=1}^N [\Psi_i(w) = 0]$ ). This phase consists of  $\text{poly}(t, \log N, \log k, d)$  bits of interaction.

**Remark:** The moreover part of Theorem 9 and Theorem 10 implies that these NP languages have zero-knowledge proofs that consist of two phases: The first phase is non-interactive, and depends only on the witness  $w$  (and not on the instance  $x$ ). In this phase the prover sends to the verifier a certain (non-interactive) commitment to his witness at hand. The second phase is interactive and is very short. In this phase the prover and verifier engage in a zero-knowledge proof that indeed the string committed to in the first phase is a valid witness for  $x$ .

These types of succinct zero-knowledge proofs may have several applications. We refer the reader to [KR06] for a list of some of these applications.

**Deriving Theorem 9 from Theorem 10:** Assume w.l.o.g. that  $k \geq \log n$  (otherwise, determining whether  $x \in L$  can be done in polynomial time). Fix a parameter  $t = t(n) \leq n$ , and assume the existence of a (one-way) function  $f : \{0, 1\}^t \rightarrow \{0, 1\}^t$  that can be computed in time  $\text{poly}(n)$  and cannot be inverted by  $\text{poly}(n)$ -time adversaries (except with  $\text{negl}(n)$  probability). Let  $L = \{x : \exists w \text{ s.t. } \mathcal{R}_L(x, w) = 1\}$  be an NP language, such that  $\mathcal{R}_L$  is a constant depth Boolean formula (over the gates  $\{\neg, \vee, \wedge, \oplus\}$ ). Recall that for every  $x \in \{0, 1\}^n$ , by applying Lemma 7.1 (in Section 7) with the Boolean formula  $\mathcal{R}_L(x, \cdot) : \{0, 1\}^k \rightarrow \{0, 1\}$  and with  $\delta = 2^{-(t+1)}$ , one can approximate  $\mathcal{R}_L(x, \cdot)$  (with error  $\leq 2^{-(t+1)}$ ) by an arithmetic formula of size  $\text{poly}(n)$  and of degree  $d \leq \text{poly}(\log n, t)$ . Fix  $N = 1$ ,  $k = |w|$ , and  $d$  as above. Fix any  $(x, w)$  such that  $\mathcal{R}_L(x, w) = 1$ . A prover, who has  $w$  as an additional input, can prove that  $x \in L$ , by emulating the zero-knowledge proof for  $\text{SAT}_{N,k,d}$ , given by Theorem 10, as follows. We use Theorem 10 with the assumption that there exists a (one-way) function  $f : \{0, 1\}^{t+1} \rightarrow \{0, 1\}^{t+1}$  that can be computed in time  $\text{poly}(n)$  and cannot be inverted by  $\text{poly}(n)$ -time adversaries (except with  $\text{negl}(n)$  probability). This assumption follows trivially from our assumption (on the existence of a (one-way) function  $f : \{0, 1\}^t \rightarrow \{0, 1\}^t$  with similar properties), and results with soundness parameter  $s \leq 2^{-(t+1)}$ .

**Phase 1.** The prover runs the first phase of the zero-knowledge proof for  $\text{SAT}_{N,k,d}$ , given by Theorem 10, with his witness at hand  $w$ . Namely, he commits to his witness  $w = (w_1, \dots, w_k)$ , as in the first phase of the zero-knowledge proof given by Theorem 10. We use here the fact that this phase depends only on the witness  $w$  and not on the instance.

**Phase 2.** The prover and verifier approximate the Boolean formula  $\mathcal{R}_L(x, \cdot) : \{0, 1\}^k \rightarrow \{0, 1\}$  by a low degree arithmetic formula  $\Psi$ , using Lemma 7.1 with  $\delta = 2^{-(t+1)}$ . To this end, the verifier chooses  $\text{poly}(\log n, t)$  random bits, and sends these random bits to the prover. Then the prover and verifier (separately) run in time  $\text{poly}(n)$ , and compute an arithmetic formula  $\Psi$ , of size  $\text{poly}(n)$  and of degree  $d \leq \text{poly}(\log n, t)$ , such that

$$\Pr[\Psi(w) \neq \mathcal{R}_L(x, w)] \leq 2^{-(t+1)}.$$

Then, the prover and verifier run the second phase of the zero-knowledge proof for  $\text{SAT}_{N,k,d}$ , given by Theorem 10 (with soundness parameter  $s = 2^{-(t+1)}$ ), for proving that the message

committed to in the first phase is a witness for  $1 - \Psi \in \text{SAT}_{N,k,d}$ . Thus, the second phase consists of  $\text{poly}(t, \log n)$  bits of interaction.

Theorem 10, together with Lemma 7.1, implies that this protocol is zero-knowledge, with completeness parameter  $c \geq 1 - 2^{-t}$  and soundness parameter  $s \leq 2^{-(t+1)} + 2^{-(t+1)} = 2^{-t}$ , as desired.

**Proof of Theorem 10:** Fix a parameter  $t = t(n) \leq n$ , and assume the existence of a (one-way) function  $f : \{0, 1\}^t \rightarrow \{0, 1\}^t$  that can be computed in time  $\text{poly}(n)$  and cannot be inverted by  $\text{poly}(n)$ -time adversaries (except with  $\text{negl}(n)$  probability). This implies that there exists a bit commitment scheme

$$\text{com} : \{0, 1\} \times \{0, 1\}^t \rightarrow \{0, 1\}^{\text{poly}(t)}$$

that uses  $t$  bits of randomness, is perfectly binding, and is computationally hiding against  $\text{poly}(n)$ -time adversaries. (See [G01] for the definition of a commitment scheme and for the proof method).

Theorem 6 implies that for every  $s \geq 2^{-n}$ , there exists an interactive PCP for the NP language  $\text{SAT}_{N,k,d}$  with parameters  $(p, q, \ell, c, s)$ , where  $p = \text{poly}(k, d)$ ,  $q = \text{poly}(\log \frac{1}{s})$ ,  $\ell = \text{poly}(\log N, \log k, d, \log \frac{1}{s})$ , and  $c = 1$ . Fix  $s \stackrel{\text{def}}{=} \frac{2^{-t}}{2} \geq 2^{-(n+1)}$ . We use the interactive PCP from Theorem 6, with soundness parameter  $s$ ,<sup>17</sup> to construct a zero-knowledge proof  $(P_{\text{ZK}}, V_{\text{ZK}})$  for  $\text{SAT}_{N,k,d}$ .

**Input:** Both the prover  $P_{\text{ZK}}$  and the verifier  $V_{\text{ZK}}$  take as input  $N$  arithmetic formulas  $\Psi_1, \dots, \Psi_N : \{0, 1\}^k \rightarrow \{0, 1\}$  of syntactic degree  $\leq d$ . The prover  $P_{\text{ZK}}$  takes an additional input  $w = (w_1, \dots, w_k) \in \{0, 1\}^k$ , such that

$$\bigwedge_{i=1}^N [\Psi_i(w_1, \dots, w_k) = 0].$$

**Phase 1.** The prover  $P_{\text{ZK}}$  generates the string  $\pi$  of size  $p = \text{poly}(k, d)$ , as in Theorem 6. Recall that, according to Theorem 6,  $\pi$  depends only on  $w$  and on the parameter  $d$ , and not on the instance  $\Psi_1, \dots, \Psi_N$ . He sends  $\text{com}(\pi_1), \dots, \text{com}(\pi_p)$  to the verifier  $V_{\text{ZK}}$ , where  $\text{com}$  is the commitment scheme specified above. The message sent is of size  $\text{poly}(t, p) = \text{poly}(t, k, d)$ .

**Phase 2.** According to Theorem 6, the interactive PCP verifier (given by Theorem 6) can partition his work into two phases. In the first phase he runs in time  $\leq \text{poly}(|\Psi_1| + \dots + |\Psi_N|)$ , and generates a string  $h$  of size  $\leq \text{poly}(\log N, \log k, d, \log \frac{1}{s})$ . This phase depends only on  $\Psi_1, \dots, \Psi_N$  and on the parameters (and on the randomness of the verifier) and does not depend on the oracle  $\pi$  or on the interaction with the prover. In the second phase, which is the interactive phase, the messages sent by the verifier, and the oracle queries, depend only on  $h$  (and not on  $\Psi_1, \dots, \Psi_N$ ), and are independent of the messages sent by the prover and the oracle answers. Moreover, the verifier in this phase runs in time  $\leq \text{poly}(\log N, \log k, d, \log \frac{1}{s})$ .

1. The verifier  $V_{\text{ZK}}$  computes the string  $h$ , as in the first phase of the interactive PCP given by Theorem 6.
2. The prover  $P_{\text{ZK}}$  and verifier  $V_{\text{ZK}}$  run the interactive protocol of the interactive PCP given by Theorem 6, with the following difference: The prover  $P_{\text{ZK}}$ , rather than sending his messages “in the clear,” will commit to all his messages. Namely, if the interactive PCP protocol consists of a transcript of the form:

$$(r_1, m_1, r_2, m_2, \dots, r_l, m_l),$$

---

<sup>17</sup>By padding the instance, Theorem 6 also holds with  $s \geq 2^{-(n+1)}$ , without changing any of the other parameters.

then in the (zero-knowledge) protocol, the transcript will be of the form:

$$(r_1, \text{com}(m_1), r_2, \text{com}(m_2), \dots, r_l, \text{com}(m_l)).$$

Since the verifier's messages  $r_1, \dots, r_l$  are determined by  $h$ , and do not depend on the interaction,  $V_{\text{ZK}}$  does not need to “know” the messages  $m_1, \dots, m_l$  (or the answers to his oracle queries) in order to generate  $r_1, \dots, r_l$ .

3. The verifier  $V_{\text{ZK}}$  sends the string  $h$  to the prover. Note that the string  $h$  determines the  $q$  oracle queries  $i_1, \dots, i_q \in [p]$  of the interactive PCP verifier.
4. Finally, the prover  $P_{\text{ZK}}$  gives the verifier a zero-knowledge proof that

$$(h, \text{com}(\pi_{i_1}), \dots, \text{com}(\pi_{i_q}), r_1, \text{com}(m_1), \dots, r_l, \text{com}(m_l))$$

corresponds to an accepting proof in the underlying interactive PCP. Namely, the prover  $P_{\text{ZK}}$  gives the verifier  $V_{\text{ZK}}$  a zero-knowledge proof that the interactive PCP verifier, who chose the string  $h$ , accepts the proof that consists of the transcript corresponding to

$$(r_1, \text{com}(m_1), \dots, r_l, \text{com}(m_l))$$

and the oracle answers corresponding to

$$(\text{com}(\pi_{i_1}), \dots, \text{com}(\pi_{i_q})).$$

$P_{\text{ZK}}$  uses a zero-knowledge proof that has the following properties: its soundness is  $s$  (recall that  $s \stackrel{\text{def}}{=} \frac{2^{-t}}{2}$ ), its completeness is  $c = 1$ , it is zero-knowledge against  $\text{poly}(n)$ -time adversaries, and its communication complexity is at most

$$\text{poly}(t, \log N, \log k, d).$$

Such a proof exists since the statement that we wish to prove is in

$$\text{NTIME}(\text{poly}(t, \log N, \log k, d)).$$

This follows from the fact that the instance

$$(h, \text{com}(\pi_{i_1}), \dots, \text{com}(\pi_{i_q}), r_1, \text{com}(m_1), \dots, r_l, \text{com}(m_l))$$

is of size  $\leq \text{poly}(t, \log N, \log k, d)$ , and the running time of the interactive PCP verifier in the second phase is at most  $\text{poly}(t, \log N, \log k, d)$ . Thus, using the commitment scheme  $\text{com}$  as above, we can easily construct a proof that is zero-knowledge against  $\text{poly}(n)$ -time adversaries, with communication complexity at most  $\text{poly}(t, \log N, \log k, d)$ .

5. The verifier accepts if and only if he accepts this proof.

The fact that this protocol is zero-knowledge follows from the fact that the commitment scheme  $\text{com}$  is computationally hiding against  $\text{poly}(n)$ -time adversaries, and from the fact that the protocol in Step 4 is zero-knowledge against  $\text{poly}(n)$ -time adversaries. The perfect completeness of this protocol follows from the perfect completeness of the zero-knowledge protocol in Step 4, and from the perfect completeness of the underlying interactive PCP. The soundness parameter of this protocol is  $2s = 2^{-t}$ . This follows from the soundness parameter  $s$  of the zero-knowledge protocol in Step 4, and from the soundness parameter  $s$  of the underlying interactive PCP (and from the fact that the commitment scheme  $\text{com}$  is perfectly binding). The fact that the prover is efficient (assuming that he is given a witness) again follows from the fact that the prover in the interactive PCP is efficient, and from the fact that the prover in the zero-knowledge proof in Step 4 is efficient. ■



## References

- [ALMSS92] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof Verification and Hardness of Approximation Problems. In *FOCS 1992: 14-23*. Also in *J. ACM 45(3): 501-555 (1998)*.
- [AS92] S. Arora, and S. Safra: Probabilistic Checking of Proofs: A New Characterization of NP. In *FOCS 1992: 2-13*. Also in *J. ACM 45(1): 70-122 (1998)*.
- [AS97] S. Arora, and M. Sudan. Improved Low-Degree Testing and its Applications. In *STOC 1997: 485-495*. Also in *Combinatorica 23(3): 365-426 (2003)*.
- [BFL90] L. Babai, L. Fortnow, and C. Lund. Non-Deterministic Exponential Time has Two-Prover Interactive Protocols. In *FOCS 1990: 16-25*. Also In *Computational Complexity 1: 3-40 (1991)*.
- [BGGHKMR88] M. Ben-Or, O. Goldreich, S. Goldwasser, J. Hastad, J. Kilian, S. Micali, P. Rogaway. Everything Provable is Provable in Zero-Knowledge. In *CRYPTO 1988: 37-56*.
- [BGKW88] M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson. Multi-Prover Interactive Proofs: How to Remove Intractability Assumptions. In *STOC 1988: 113-131*.
- [BM88] L. Babai, and S. Moran. Arthur-Merlin Games: A Randomized Proof System, and a Hierarchy of Complexity Classes. *J. Comput. Syst. Sci. 36(2): 254-276 (1988)*.
- [B93] R. Beigel. The Polynomial Method in Circuit Complexity. In *Structure in Complexity Theory Conference 1993: 82-95*.
- [DFKRS99] I. Dinur, E. Fischer, G. Kindler, R. Raz, and S. Safra. PCP Characterizations of NP: Towards a Polynomially-Small Error-Probability. In *STOC 1999: 29-40*.
- [FGLSS91] U. Feige, S. Goldwasser, L. Lovasz, S. Safra, and M. Szegedy. Interactive Proofs and the Hardness of Approximating Cliques. In *FOCS 1991: 2-12*. Also in *J. ACM 43(2): 268-292 (1996)*.
- [FL92] U. Feige, and L. Lovasz. Two-Prover One-Round Proof Systems: Their Power and Their Problems (Extended Abstract) In *STOC 1992: 733-744*.
- [FS07] L. Fortnow, and R. Santhanam. Infeasibility of Instance Compression and Succinct PCPs for NP. *Electronic Colloquium on Computational Complexity, Report TR07-096 (2007)*.
- [GKR07] S. Goldwasser, Y. T. Kalai, and G. Rothblum. Delegating Computation: Interactive Proofs for Mortals. Manuscript 2007.
- [GMR85] S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. In *STOC 1985: 291-304*. Also in *SIAM Journal on Computing, 18(1):186-208, 1989*.
- [GMW86] O. Goldreich, S. Micali, and A. Wigderson. Proofs that Yield Nothing But Their Validity or All Languages in NP Have Zero-Knowledge Proof Systems. In *J. ACM 38(3):691-729 (1991)*.
- [G01] O. Goldreich. Foundations of Cryptography: Volume 1 – Basic Tools. *Cambridge University Press, 2001*.

- [HN06] H. Harnik, and M. Naor. On the Compressibility of NP instances and Cryptographic Applications. In *FOCS 2006*: 719-728.
- [IKOS07] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Zero-Knowledge from Secure Muliparty Computation. In *STOC 2007*: 21-30.
- [KR06] Y. T. Kalai, and R. Raz. Succinct Non-Interactive Zero-Knowledge Proofs with Preprocessing for LOGSNP. In *FOCS 2006*: 355-366.
- [KR07] Y. T. Kalai, and R. Raz. Probabilistically Checkable Arguments. Manuscript in preparation.
- [K92] J. Kilian. A note on efficient zero-knowledge proofs and arguments. In *STOC 1992*: 723-732.
- [LFKN90] C. Lund, L. Fortnow, H. J. Karloff, and N. Nisan. Algebraic Methods for Interactive Proof Systems. In *FOCS 1990*: 2-10. Also in *J. ACM 39(4)*: 859-868 (1992).
- [MR05] D. Moshkovitz, and R. Raz. Sub-Constant Error Low Degree Test of Almost Linear Size. In *STOC 2006*: 21-30. Also appears on *(ECCC)(086)*: (2005).
- [MR06] D. Moshkovitz, and R. Raz. Sub-Constant Error PCP of Almost Linear Size. (Manuscript 2006).
- [M94] S. Micali. CS Proofs (Extended Abstracts). In *FOCS 1994*: 436-453.
- [RS97] R. Raz, and S. Safra. A Sub-Constant Error-Probability Low-Degree Test, and a Sub-Constant Error-Probability PCP Characterization of NP. In *STOC 1997*: 475-484.
- [R87] A. Razborov. Lower Bounds for the Size of Circuits of Bounded Depth with Basis  $\{\wedge, \oplus\}$ . In *Math. Notes of the Academy of Science of the USSR*: 41(4) : 333-338 (1987).
- [R05] R. Raz. Quantum Information and the PCP Theorem. In *FOCS 2005*: 459-468.
- [S92] A. Shamir. IP=PSPACE. In *J. ACM 39(4)*: 869-877 (1992).
- [S87] R. Smolensky. Algebraic Methods in the Theory of Lower Bounds for Boolean Circuit Complexity. In *STOC 1987*: 77-82.