# Quantifying Knowledge Complexity*

Oded Goldreich† Erez Petrank‡

July 17, 1997

**Abstract**

One of the many contributions of the paper of Goldwasser, Micali and Rackoff is the introduction of the notion of knowledge complexity. Knowledge complexity zero (also known as zero-knowledge) have received most of the attention of the authors and all the attention of their followers. In this paper, we present several alternative definitions of knowledge complexity and investigate the relations between them.

*An extended abstract of this paper appeared in *the 32nd Annual IEEE Symposium on the Foundations of Computer Science* (*FOCS91*) held in San Juan, Puerto Rico, October 1991.

†Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel. E-mail: oded@wisdom.weizmann.ac.il.

‡Computer Science Department, Technion – Israel Institute of Technology, Haifa 32000, Israel. E-mail: erez@cs.technion.ac.il.

# 1 Introduction

One of the many contributions of the seminal paper of Goldwasser, Micali and Rackoff [18] is the introduction of the notion of knowledge complexity. Knowledge complexity is intended to measure the computational advantage gained by interaction. Hence, something that can be obtained without interaction is not considered knowledge. The latter phrase is somewhat qualitative and supplies the intuition underlying the definition of zero-knowledge (i.e., knowledge complexity zero) given in [18]. Quantifying the amount of knowledge gained by interaction, in case it is not zero, is more problematic.[1] We stress that the definition of zero-knowledge *does not* depend on the formulation of *the* AMOUNT *of knowledge gained* since this definition addresses the case in which NO *knowledge is gained.*

## 1.1 Motivation for the study of Knowledge Complexity

Whatever a party can compute in solitude is a function of the information it has and its computing power. However, when two (or more) parties interact their individual computing abilities may increase as a result of information they receive from other parties. Knowledge complexity is intended to capture this increase in computing ability. Thus, *knowledge complexity* is a fundamental measure of interaction between parties, and it differs from other measures of interaction such as *information entropy* [27, 9] and *communication complexity* [28, 24]. The following examples may help to illustrate what we mean. In all these examples we assume that Bob is restricted to probabilistic polynomial-time (in the parameter $n$), whereas no computation restrictions are placed on Alice. We also assume throughout this discussion that integer factorization is an infeasible task (i.e., cannot be performed in probabilistic polynomial-time).

**Example 1** Suppose that Alice uniformly selects a string $r \in \{0,1\}^n$ and sends it to Bob. From an information theoretic point of view, Bob has received $n$ bits (of maximum entropy) and the same holds with respect to the point of view of communication complexity. However, from the computational complexity point of view Bob has received nothing as he could have generated $r$ by himself. Thus, Alice's message carries knowledge complexity zero (i.e., is zero-knowledge).

**Example 2** Suppose that Alice just sends Bob the string $1^n$. Still, $n$ bits are communicated but both information theory and computational complexity view the interaction as being of zero contents. Again, Alice message has knowledge complexity zero since again Bob might have produced the received message in solitude.

**Example 3** Suppose that Alice sends Bob the prime factorization of an $n$-bit composite number which is given to both of them from the outside. Again, information theory says that Bob has gained nothing from Alice's message (since the prime factorization is determined by the composite number known to Bob). However, from the point of view of computational complexity Bob has *gained a lot*: he can now perform many tasks which he could not have done before receiving the factorization (e.g., extract square roots modulo the composite number).

The above examples are well known and so is their analysis which has been focused at the qualitative question of whether Alice's behavior (in these examples) is zero-knowledge or not. However, as is the case in information theory and communication complexity, the spectrum of

[1] It seems that, in general, quantitative notions are harder to handle than qualitative ones.

possible behaviors is wider than being binary (i.e., carrying zero-knowledge or not). The following examples are aimed at demonstrating this point.

**Example 4** Both Alice and Bob are given a composite number, denoted $N$, which is the product of two primes each congruent to 3 mod 4. Let $QNR_N^+$ denote the set of quadratic non-residues mod $N$ with Jacobi Symbol $+1$. Recall that one fourth of the elements of $Z_N^*$ are in $QNR_N^+$ and that it is considered infeasible to distinguish elements of $QNR_N^+$ from quadratic residues mod $N$ [17]. Suppose that Alice uniformly selects a $y \in QNR_N^+$ and sends it to Bob. It seems that Bob has gained some knowledge (as we don't know how to uniformly sample $QNR_N^+$ in polynomial-time when only given $N$). On the other hand, it seems that Bob did not gain much knowledge. In particular, he still cannot factor $N$ (i.e., with the help of $y$). To see that an element of $QNR_N^+$ adds little knowledge towards the factorization of $N$ (or any other NP-task) note that Bob can uniformly select a residue with Jacobi Symbol $+1$ mod $N$. Suppose that Bob does so and let $y$ denote the residue produced by Bob. With probability $1/2$, $\quad y \in QNR_N^+$ (and otherwise $y$ is a quadratic residue modulo $N$). Bob does not know whether $y$ is in $QNR_N^+$ but if elements in $QNR_N^+$ facilitate the factorization of $N$ then Bob may try to factor $N$ and succeed whenever $y$ is indeed in $QNR_N^+$ (which as we said happens with probability $1/2$). Thus, the message sent by Alice seems to yield knowledge but not much (especially when compared to Alice's message in Example 3).

Another point worth noting is that repeated executions of Alice's step do not increase the knowledge gained by Bob. This holds since given any $y \in QNR_N^+$, Bob can easily generate (by himself!) uniformly distributed elements of $QNR_N^+$ (e.g., by uniformly selecting $r \in Z_N^*$ and outputting $y \cdot r^2 \pmod N$).

**Example 5** Let $N$ be as in Example 4. Suppose that Alice agrees to provide Bob with the least significant bit of the square root (mod $N$) of any quadratic residue mod $N$ of Bob's choice. By [20, 3] such an answer (by Alice) does yield knowledge to Bob and furthermore $|N|$ answers of this form allow Bob to factor $N$. Thus, although each answer yields little knowledge (as can be argued analogously to Example 4), many answers yield substantial knowledge.

Examples 3, 4 and 5 demonstrate that there is more to knowledge complexity than merely determining whether a protocol is zero-knowledge or not. Following Goldwasser, Micali and Rackoff [18], we suggest that the knowledge gained by interaction can be quantified. The analogy to information theory and communication complexity is telling: none of these stops at a binary distinction between zero and positive.

Goldwasser, Micali and Rackoff have suggested to characterize languages according to the knowledge complexity of their interactive proof systems [18]. The lowest class consists of languages having knowledge complexity zero. This class, also known as zero-knowledge, has received much attention in recent years. The following example may serve as a teaser for the low (non-zero) levels of the knowledge complexity hierarchy:

**Example 6** *composing zero-knowledge languages.* Consider, for example, the UNION of two languages each having a (perfect) zero-knowledge interactive proof system. One can certainly prove membership in the union by proving membership in one of these languages, but this, in general, seems to leak some knowledge (yet not much). Likewise, consider the language consisting of pairs so that exactly one of the elements in the pair belongs to a specific language having a (perfect) zero-knowledge proof system. These composed languages are not known to be (perfect) zero-knowledge (in general) and yet seem to have low perfect knowledge-complexity (especially when compared to the perfect knowledge-complexity of PSPACE-complete languages).

To summarize, we believe that the concept of knowledge complexity is a fundamental one and that it may play an important role in Complexity Theory. Before this can happen, an adequate definition of knowledge complexity has to be supplied. In this paper we explore several alternative ways of defining knowledge complexity and investigate the relations between them. But before we start, let us recall some of basic notions and frameworks regarding zero-knowledge.

## 1.2 Background on zero-knowledge

Loosely speaking, an interactive proof system for a language $L$ is a two-party protocol, by which a powerful *prover* can "convince" a probabilistic polynomial time *verifier* of membership in $L$, but will fail (with high probability) when trying to fool the verifier into "accepting" non-members. An interactive proof is called zero-knowledge if the interaction of any probabilistic polynomial time machine with the predetermined prover, on common input $x \in L$, can be "simulated" by a probabilistic polynomial time machine (called the *simulator*). Thus, the definition considers two types of probability ensembles, where each ensemble associates a distribution to each $x \in L$. The two different distribution associated to each $x \in L$ are:

1. The distribution of the view of a probabilistic polynomial time machine with the prover on common input $x \in L$.
2. The output distribution of a probabilistic polynomial time machine (the *simulator*) on the same input $x$.

It is required that for every distribution ensemble of type (1), there is a distribution ensemble of type (2) such that these two distribution ensembles are similar. Similarity is interpreted in three possible ways yielding three different definitions of zero-knowledge.

1. The most conservative interpretation is that the ensembles are identical. The resulting definition is called *perfect* zero-knowledge. An example of a language having a perfect zero-knowledge interactive proof is Quadratic Non-Residuosity [19].
2. Slightly more liberal is the requirement that the ensembles are statistically close, namely that their variation distance (Norm-1 difference) is negligible (i.e., smaller than any polynomial fraction in the length of the common input). The resulting definition is called *statistical* (or *almost perfect*) zero-knowledge. For example, the results of Fortnow [10] and of Aiello and Håstad [2] on the "complexity of zero-knowledge" refer to this definition.
3. Most liberal is the requirement that the ensembles are indistinguishable by all probabilistic polynomial time tests. The resulting definition is called *computational* zero-knowledge. For example, the result of [13] asserting that "all languages in NP have zero-knowledge proofs provided that commitment schemes exist" refers to this definition.

## 1.3 Defining Knowledge Complexity

Unless otherwise indicated, the following discussion refers to the definitions of knowledge complexity in which the simulated conversations are close to the real one in the statistical sense. Namely, we consider the hierarchy of knowledge complexity extending statistical zero-knowledge (as the zero level). Here we consider knowledge complexity in the context of interactive proof systems. The actual definitions apply to any pair of interactive machines.

**A first attempt.** An attempt to formalize the "amount of knowledge" (in case it is not zero) has appeared in the preliminary version of [18] but was omitted from the later version of this work [19] since the authors themselves found it inadequate (Micali, private communication). By the preliminary formulation of [18] the knowledge complexity of an interactive proof $(P, V)$ is said to be $k(|x|)$ if there exists a simulator which can generate a distribution $M(x)$ such that the variation distance[2] of $M(x)$ and $(P, V)(x)$ is bounded above by $1 - 2^{-k(|x|)} + |x|^{-c}$ for all constants $c > 0$ and sufficiently long $x$'s. Hence, any prover which with probability $\frac{1}{2}$ sends nothing, leaks (by this definition) at most 1 bit of knowledge. In particular, a prover that with probability $\frac{1}{2}$ reveals a Hamiltonian path in the input (Hamiltonian) graph (and otherwise reveals nothing), is considered to leak only one bit of knowledge (contradicting our feeling that this prover gives away a lot). However, the same analysis applies to a prover which merely tell the verifier whether the input graph is Hamiltonian or not. Thus, this measure does not distinguish between these provers and this contradicts our feelings that the former prover gives away much more knowledge. Furthermore, by this definition, all protocols have knowledge complexity bounded by $k(|x|) = \frac{\log(|x|)}{o(1)}$; the reason being that one can simulate them by any distribution, since the variation distance (of any distribution) towards the real interaction is at most 1 which, in turn, is smaller than $1 - 2^{-k(|x|)} + |x|^{-O(1)} = 1 - |x|^{1/o(1)} + |x|^{-O(1)}$. In particular, it follows that all languages in the class $\mathcal{IP}$ (i.e., all languages having interactive proof systems) have knowledge complexity $\frac{\log(|x|)}{o(1)}$. This contradicts our feeling that one may give away more knowledge; see below. We mention that, prior to our work, the notion of knowledge complexity (apart from zero-knowledge) has not been investigated any further.

**A suggested criteria.** Our search for a definition of knowledge complexity is guided by the intuitive discussion in Subsection 1.1. In particular, we postulate that the knowledge complexity of a protocol must be bounded above by its communication complexity. We also believe that this upper bound should be obtainable, by some protocols, for every possible value of the communication complexity. Thus, knowledge complexity should range between zero and polynomial.

We suggest several definitions of knowledge complexity and discuss them in light of the above criteria. Our definitions utilize well-established frameworks such as oracle machines, communication complexity, conditional probability spaces and machines which accepts hints. Among the definitions of knowledge complexity presented below, we find the following most intuitive.

**Knowledge as communication in an alternative interaction (or simulation with the help of an oracle).** A prover $P$ is said to yield at most $k(|x|)$ bits of knowledge if whatever can be efficiently computed through an interaction with $P$ on input $x$ (in the language), can also be efficiently computed on input $x$ through an interaction with an alternative machine which sends at most $k(|x|)$ bits. Hence, the computational advantage gained by the interaction with the prover who may send much more than $k(|x|)$ bits can be simulated by an interaction with a machine which only sends $k(|x|)$ bits. In some sense, this approach reduces knowledge complexity to communication complexity. Clearly, knowledge complexity as defined here is bounded above by the communication complexity and we show (in Section 5) that the bound may be met in some cases. We note that, without loss of generality, the "knowledge-giving-machine" can be made memoryless and deterministic, by supplying it with all previous messages and with coin tosses. Hence, the "knowledge-giving-machine" is merely an oracle (and we may think of the simulation as being performed by an oracle machine and count the number of its binary queries). We present three

[2]Recall that the *variation distance* between the random variables $Y$ and $Z$ is $\frac{1}{2}\sum_{\alpha} |\mathrm{Prob}(Y = \alpha) - \mathrm{Prob}(Z = \alpha)|$.

variants of the above definition. These variants are analogous to common variants of probabilistic communication complexity. The most conservative (or "strict") approach is to consider the worst-case number of bits communicated on an input $x$. The most liberal approach is to consider the average case (the average is taken only on the coin-tosses of the simulator, the input $x$ is fixed). In between and (in light of our results) much closer to the worst-case variant is a relaxation of the worst-case variant in which the simulator is only required to produce output with probability at least one half (i.e., this is a Las Vegas variant). This last variant is hereafter referred to as *the* oracle definition of knowledge complexity.

The above definition corresponds to our intuition that Alice behavior in Examples 4 and 5 gives away very little knowledge (in each message). This seem clear with respect to Example 5 where only one bit is communicated and in Example 4 (when one uses the intermediate oracle definition).[3] On the other hand, Alice's behavior in Example 3 as well as repeated executions of the protocol in Example 5 seem to have knowledge complexity which grows with the length of the prime factors. This corresponds to our intuition that a lot of knowledge may be gained by Bob in these cases.

**Knowledge as the measure of a good conditional probability space.** An alternative approach to knowledge complexity is to provide the simulator (of the interaction between the prover and the verifier) no help, but rather relax the requirement concerning its output distribution. Instead of requiring, as in the zero-knowledge case, that the output distribution of the simulation be similar to the distribution of the "real interaction", we only require that former distribution contains a (*good*) *subspace* which is similar to the distribution of the "real interaction". Knowledge complexity is defined to reflect the density of this good subspace (in the output distribution). Specifically, knowledge complexity is defined as (minus) the logarithm of the density of this good subspace in the entire probability space (i.e., the output distribution). Again, knowledge complexity as defined here is bounded above by the communication complexity and we show (in Section 5) that the bound may be met in some cases. Also note that Alice's behavior in Example 4 is clearly of knowledge complexity 1 according to the definition here. Interestingly, the definition of knowledge complexity as a "logarithm of the good fraction" agrees with the informal discussion in [18] (although the formal definition presented there was different – see above). In fact, Micali (private communication) has independently discovered the "fraction" definition.

We show that knowledge-complexity as per the fraction measure approximates (up to an additive constant) the oracle measure of knowledge-complexity. The fact that these intuitive but seemingly different approaches to knowledge-complexity yields very close measures adds to our feeling that these definitions are the adequate ones.

**Knowledge as the length of a hint.** Our last interpretation of knowledge-complexity is as the length of the shortest hint which allows an efficient machine to simulate the real interaction. Unlike in the oracle approach to knowledge complexity, this hint is a predetermined function of the input and, in particular, does not depend on the coin-tosses of the simulator (or even the verifier). Hence, the amount of knowledge (in the hint sense) leaked by two executions of the same protocol on the same input, *always* equals the amount of knowledge leaked by a single execution. We note that the hint-length measure was suggested in [6], and seems adequate in the information theoretic model discussed there.

We have several reasons for objecting to the last measure of knowledge-complexity (i.e., the hint length measure). Firstly, knowledge complexity by the hint measure is not bounded by the

[3] Using the strict oracle measure of knowledge complexity, we only establish a super-logarithmic upper bound; see results below.

communication complexity; this is shown explicitly in Section 5. In particular, we present a protocol in which Alice sends a single bit but no polynomial-length hint can enable to compute this bit and so simulate this protocol (let alone in polynomial-time). Furthermore, we conjecture that the protocols in Examples 4 or 5 (even when executed once) have knowledge complexity which grows with the length of the prime factors, which puts them in the same category as the protocol of Example 3 (whereas our intuition is that the latter yields much more knowledge). Nevertheless, the hint measure can be used to satisfactory bound the knowledge complexity of the languages in Example 6 (i.e., 1-bit long hint suffices there). This may raise hopes that, although too conservative as a measure for protocols, the hint measure may be adequate for defining knowledge complexity classes of languages (i.e., by considering the knowledge complexity of the cheapest interactive proof). These hopes should be abandoned if one believes, as us, that all languages in $\mathcal{IP}$ must have at most polynomial knowledge-complexity: In Section 6 we extend the work of Aiello and Håstad [2], who in turn follow Fortnow's ideas [10], showing that languages having polynomial knowledge-complexity in the hint sense are in $\mathcal{AM}[2]$.[4] Thus, languages having polynomial knowledge-complexity by the hint measure are unlikely to contain all of $\mathcal{IP}$.

### 1.4 Relating the various definitions of Knowledge Complexity

In order to summarize our results concerning the relations between the various definitions, we present the following *unsound* notations.[5] Let $\mathrm{kc}^{\mathrm{strict}}_{\mathrm{oracle}}(\Pi), \overline{\mathrm{kc}}_{\mathrm{oracle}}(\Pi)$ and $\mathrm{kc}^{1/2}_{\mathrm{oracle}}(\Pi)$ denote the knowledge complexity of a protocol $\Pi = (P, V)$ according to the strict, average and "intermediate" (i.e., output with probability 1/2) variants of the oracle approach. Likewise, $\mathrm{kc}_{\mathrm{fraction}}(\Pi)$ and $\mathrm{kc}_{\mathrm{hint}}(\Pi)$ denote the knowledge complexity of $\Pi$ according to the "fraction" and "hint" approaches. In the following lines we informally summarize our results concerning the relations between the various definitions:

**1.** *Obvious inclusions between the various oracle measures:* For every protocol $\Pi$,

$$\overline{\mathrm{kc}}_{\mathrm{oracle}}(\Pi) - 2 \le \mathrm{kc}^{1/2}_{\mathrm{oracle}}(\Pi) \le \mathrm{kc}^{\mathrm{strict}}_{\mathrm{oracle}}(\Pi) \le \mathrm{kc}_{\mathrm{hint}}(\Pi)$$

**2.** *Closeness of the oracle and the fraction measures:* For every protocol $\Pi$,

$$\mathrm{kc}_{\mathrm{fraction}}(\Pi) - 1 \le \mathrm{kc}^{1/2}_{\mathrm{oracle}}(\Pi) \le \mathrm{kc}_{\mathrm{fraction}}(\Pi) + 4$$

**3.** *The strict oracle measure versus the oracle measure:* For every protocol $\Pi$ and any unbounded function $g : N \to N$,

$$\mathrm{kc}^{\mathrm{strict}}_{\mathrm{oracle}}(\Pi) \le \mathrm{kc}^{1/2}_{\mathrm{oracle}}(\Pi) + \log(\log(|x|)) + g(|x|)$$

On the other hand, for any $c > 0$ and for any polynomial $p : N \to N$, there exists a protocol $\Pi$ such that

$$\mathrm{kc}^{1/2}_{\mathrm{oracle}}(\Pi) = p(|x|) \text{ and } \mathrm{kc}^{\mathrm{strict}}_{\mathrm{oracle}}(\Pi) \ge \mathrm{kc}^{1/2}_{\mathrm{oracle}}(\Pi) + \log(\log(|x|)) + c.$$

[4] We also show that languages having logarithmic knowledge-complexity in the hint sense are in co$\mathcal{AM}[2]$.

[5] The notations below suggest that knowledge-complexity is a functional which assigns each protocol a unique function (i.e., upper bound). This is inaccurate since actually each protocol is assigned a (countable) set of functions; see Section 2. Still, we urge the reader to ignore this point for the time being, and think of the knowledge-complexity of a protocol, according to each measure, as of a single function. The formal interpretation of the results below can be derived by using the notational conventions of Section 2.

(Here and throughout the rest of the paper, all logarithms are to base 2.)

4. *The average oracle measure may be much smaller than the oracle measure:* For every polynomial $p(|x|)$, there exists a protocol $\Pi$ such that $\overline{\mathrm{kc}}_{\mathrm{oracle}}(\Pi) \leq \frac{1}{p(|x|)}$ and $\mathrm{kc}^{1/2}_{\mathrm{oracle}}(\Pi) \geq p(|x|)$.

5. *The hint measure may be much larger than the oracle measure:* There exists a protocol $\Pi$ such that $\mathrm{kc}^{\mathrm{strict}}_{\mathrm{oracle}}(\Pi) = 1$ and $\mathrm{kc}_{\mathrm{hint}}(\Pi) > p(|x|)$ for any polynomial $p$.

6. *Each measure gives rise to a strict hierarchy:* For every polynomial-time computable function $k(\cdot)$, there exists an protocol $\Pi$ such that $kc(\Pi) \leq k(|x|)$ by all the measures suggested in this paper, yet it also holds that $kc(\Pi) > k(|x|) - 3$ by all these measures.

The last result asserts that the knowledge complexity hierarchy of protocols (classified by their knowledge complexity) is "rich". We remark that an analogous result for the knowledge complexity of languages would separate $\mathcal{BPP}$ from $\mathcal{PSPACE}$.

Among our proofs of the above results, we find the proof which upper bounds the fraction measure by the oracle measure (i.e., the proof of Proposition 4.3) to be the most interesting one.

### 1.5 Subsequent works

Following the conference publication of this paper, subsequent work has been done in two directions.

The first direction, pursued in [7], [15] and [26], focuses on the oracle (or, equivalently fraction) measure of knowledge complexity and is aimed at relating the knowledge complexity of languages to their computational complexity. The first step was taken by Bellare and Petrank [7] who showed that any language having a $r(\cdot)$-round interactive proof of (statistical) knowledge complexity $k(\cdot)$, where $k(n) \cdot r(n) = O(\log n)$, resides in $\mathcal{BPP}^{\mathcal{NP}}$. Subsequently, Goldreich, Ostrovsky and Petrank [15] showed that all languages that have logarithmic (statistical) knowledge complexity are in $\mathcal{BPP}^{\mathcal{NP}}$. This was done by providing a refined analysis of a procedure in [7] and by relating the hierarchies of statistical and perfect knowledge-complexity *with respect to the honest verifier*[6]. In particular it was shown how to transform an interactive proof of statistical knowledge complexity $k(\cdot)$ (w.r.t. the honest verifier) into an interactive proof of perfect knowledge complexity $k(\cdot) + O(\log(\cdot))$ (w.r.t. the honest verifier). Petrank and Tardos [26] have extended [2] and [15] and showed that languages with logarithmic knowledge complexity are in $\mathcal{AM} \cap \mathrm{co}\mathcal{AM}$. Thus, unless the polynomial time hierarchy collapses, $\mathcal{NP}$-complete languages have super-logarithmic knowledge-complexity. They also discussed the connection between the knowledge-complexity, $k(\cdot)$, of an interactive proof and its error probability,[7] $\epsilon(\cdot)$, and showed that if $\epsilon(n) < 2^{-3k(n)}$ then the language proven has to be in the third level of the polynomial time hierarchy.

The second direction, pursued by Aiello, Bellare and Venkatesan [1], focuses on knowledge-complexity under the *average* oracle measure. Firstly, they introduced a more refined definition of average knowledge-complexity and related it to the notion defined in this paper. Secondly, they showed that the perfect and statistical average knowledge-complexity hierarchies of languages are close up to a negligible additive term. This provides an alternative characterization of the class of Statistical Zero-Knowledge (i.e., as languages having negligible-on-the-average perfect knowledge-complexity). Thirdly, they showed that languages with average logarithmic knowledge-complexity reside in $\mathcal{BPP}^{\mathcal{NP}}$. This result, which suggests that as far as languages are concerned average knowledge-complexity is not very far from knowledge-complexity (in the non-average oracle sense),

[6] See Subsection 3.5.

[7] Note that reducing the error probability via repetition is not free in our context: it may increase the knowledge-complexity.

stands in contrast to our results by which there are protocols for which the average knowledge-complexity is vastly smaller than the non-average version.

### 1.6 Organization

In Section 2 we present the formal definitions of the various knowledge complexity measures. We also state some simple connections between the various measures. In Section 3 we further discuss these definitions. We proceed, in Section 4 by showing inclusion relations between the various measures. This section includes the assertion that the oracle measure and the fraction measure are equal up to an additive constant. In Section 5 we present some separating protocols, i.e., protocols for which the various measures assign different knowledge complexity functions. These protocols demonstrate why we cannot achieve tighter inclusion relations between the measures (tighter than the relations shown in Section 4). In Section 5 we also demonstrate the strictness of the hierarchy of protocols classified by their knowledge complexity. In Section 6, we discuss the hint measure of knowledge complexity and prove properties of this measure which indicate that it is an inadequate measure.

## 2 Definitions of knowledge complexity and some basic results

In this section we present the various definitions of knowledge complexity. In each of the various definitions, knowledge complexity zero coincides with the (known) concept of zero-knowledge. For sake of brevity we present only the definitions of knowledge complexity that extend statistical zero-knowledge. Analogue definitions can be derived to extend perfect zero knowledge and computational zero-knowledge. All the analogous results (presented in this paper) hold, except for the *perfect* knowledge-complexity analogue of Proposition 4.7 and the *computational* knowledge-complexity analogue of Section 6.

For simplicity we define knowledge complexity only in the context of interactive proof systems where we are interested in the knowledge given away by the prover. However, these definitions apply to any interaction and to the knowledge given away by any of the interactive parties. Also, for simplicity, we consider in all our definitions only machines (verifiers and simulators) which run in strict polynomial time. Analogue definitions and results for machines which run for expected polynomial time, can be derived (although, in some cases, the proofs are slightly more involved).

**Interactive proof systems:** The definition of interactive proof systems are quite robust under the choice of error probability, as long as it is bounded away from $1/2$. The reason being that the error probability can be easily decreased up-to an exponentially vanishing function (of the input length), by using sequential (or parallel) repetitions of the proof system. Things become less robust when zero-knowledge is involved, since the latter is not known to be preserved under parallel repetitions (see negative results in [12]). Still, zero-knowledge (w.r.t. auxiliary-input) is preserved under sequential repetitions (see [14]). However, not all measures of knowledge complexity defined below are preserved under sequential repetitions. Thus, when talking about an interactive proof of certain knowledge complexity we must specify the error probability of the system. Below, we adopt the convention by which (unless stated otherwise) the error probability of an interactive proof is a negligible function of the input. That is, $(P, V)$ is an **interactive proof system for $L$** if for some function $\epsilon \colon \mathsf{N} \to (0,1]$, where for every positive polynomial $p$, $\epsilon(n) < 1/p(n)$ holds for all but finitely many $n$'s, it holds that

Completeness: For every $x \in L$, the verifier $V$ accepts with probability at least $1 - \epsilon(|x|)$ when interacting with $P$ on common input $x$.

Soundness: For every $x \notin L$ and any prover strategy $P'$, the verifier $V$ accepts with probability at most $\epsilon(|x|)$ when interacting with $P'$ on common input $x$.

**Some Standard Notations:** Let $(P, V)(x)$ be a random variable representing $V$'s view of the interaction of $P$ and $V$ on input $x$. This includes $V$'s coin tosses and the (the transcript of) the interaction. The probability space is that of all possible outcomes of the internal coin-tosses of $V$ and $P$. We will be interested in probability ensembles which associate a distribution (equivalently, a random variable) to each $x \in L \subseteq \{0,1\}^*$. Specifically, we will consider ensembles of the form $\{(P, V')(x)\}_{x \in L}$ and $\{M'(x)\}_{x \in L}$, where $V'$ is an arbitrary polynomial-time interactive machine and $M'$ is a simulator. When we say that two ensembles, $\{D_1(x)\}_{x \in L}$ and $\{D_2(x)\}_{x \in L}$, are *statistically close* we mean that the variation distance between them is a negligible function in the length of $x$. That is, for every positive polynomial $p$ and for all sufficiently long $x$'s

$$\sum_{\alpha} |\text{Prob}(D_1(x) = \alpha) - \text{Prob}(D_2(x) = \alpha)| < 1/p(|x|)$$

**A Non-Standard Notation:** The knowledge-complexity measures defined below are upper bounds. Each measure associates with each protocol a (countable) set of upper bound functions rather than a single function.[8] This raises a problem when we want to compare two different measures. To this end we adopt the following notational convention regarding inequalities between sets of functions. All functions we consider map strings to real numbers, and by $f \leq g$ we mean that $f(x) \leq g(x)$ for all $x \in \{0,1\}^*$. For sets of functions $F$ and $G$ and a function $h$, the notation $F \leq G + h$ means $\forall g \in G \exists f \in F$ so that $f \leq g + h$. Consequently $F \geq G + h$ means $\forall f \in F \exists g \in G$ so that $f \geq g + h$ ($F \geq h$ means that $\forall f \in F$ so that $f \geq h$), and $F = h$ means $\exists f \in F$ so that $f = h$. This notational convention is consistent with the intuitive meaning of upper bounds; for example, $F \leq G$ suggests that upper bounds in $F$ are not worse than those in $G$ which indeed means that for every $g \in G$ there exists an $f \in F$ so that $f \leq g$. Thus, when considering inequalities of the type $F \leq G$, the reader may get the spirit of the statement by thinking of both $F$ and $G$ as of functions.

In the following, we shall use functions over the integers, to describe the knowledge complexity of a protocol. Some of our results require that this function is polynomial time computable in the following (liberal) sense:

**Definition 2.0** *We say that the* knowledge complexity bound, *$k : N \to N$, is* polynomial time computable *if there exists a probabilistic polynomial time algorithm that on input $1^n$ outputs $k(n)$ with probability at least $\frac{2}{3}$ (equivalently, at least $1 - 2^{-n}$).*

This definition is more liberal than the standard one, in which, the input ($n$) is given to the machine in binary representation, thus allowing the machine to run only for poly(log($n$)) number of steps.

[8] The reason is that each simulator gives rise to such an upper bound function, and it is not clear whether an infimum exists.

## 2.1 The hint measure

In the first definition, knowledge is interpreted as a function of the input $x$.

**Definition 2.1 (knowledge complexity - Hint version):** *Let $k : N \to N$ be a function over the integers. We say that an interactive proof system $\Pi = (P, V)$ for a language $L$* can be simulated using a hint of length $k$, *if for every probabilistic polynomial time verifier $V'$ there exists a probabilistic polynomial (in $|x|$) time machine $M_{V'}$* (simulator) *such that for every $x \in L$ there exists a string $h(x)$ of length at most $k(|x|)$ such that the ensembles $\{M_{V'}(x, h(x))\}_{x \in L}$ and $\{(P, V')(x)\}_{x \in L}$ are statistically close.*
*The* knowledge complexity in the hint sense *is an operator, denoted* $\mathbf{kc}_{\text{hint}}$, *which assigns each interactive proof system $\Pi$ the set of functions, denoted $k_\Pi$, so that for every $k \in k_\Pi$ the protocol $\Pi$ can be simulated using a hint of length $k$.*
The class $\mathcal{KC}_{\text{hint}}(k(\cdot))$ *is the set of languages having interactive proofs with knowledge complexity (in the hint sense) bounded above by $k(\cdot)$.*

A subtle point regarding the above definition, concerns the convention of feeding the hint to the simulator. We adopt the convention by which, the hint (a binary string) is given to the machine on a special "hint tape". The hint string is located on the left side of the tape and is padded to its right by infinitely many zeros. The alternative convention, by which the padding is by a special symbol ("blank" $\notin \{0, 1\}$) provides implicit knowledge (i.e. the length of the hint) and is not compatible with the other definitions (presented below). Nevertheless, in case the knowledge complexity bound (i.e., the function $k(\cdot)$) is polynomial-time computable, the difference between the two conventions is at most a single bit.[9] In any case, for knowledge complexity zero there is no difference.

## 2.2 The oracle measures

The second definition widens the interpretation of knowledge to a stochastic one. Namely, the knowledge gained from the protocol depends on the random coins of the machine $M$ (or the verifier $V$), rather than being fixed in advance. The formalization is by use of oracle machines. There are three variants of oracle knowledge complexity. In all versions we adopt the standard complexity theoretic convention by which each oracle query is answered by a single bit.

**Definition 2.2 (knowledge complexity - strict Oracle version):** *We say that an interactive proof $\Pi = (P, V)$ for a language $L$* can be strictly simulated using $k$ oracle queries, *if for every probabilistic polynomial time verifier $V'$ there exists a probabilistic polynomial time oracle machine $M_{V'}$ and an oracle $A$ such that:*

1. *On input $x \in L$, machine $M_{V'}$ queries the oracle $A$ at most $k(|x|)$ times.*[10]
2. *The ensembles $\{M^A_{V'}(x)\}_{x \in L}$ and $\{(P, V')(x)\}_{x \in L}$ are statistically close.*

[9] The following encoding scheme allows us to tell where the the hint string ends at the cost of only one extra bit. Encode the hint string $h$ by concatenating the bit 1 to its end. Namely, $h$ is encoded as $h \circ 1$. Decoding relies on the fact that we have an upper bound on the length of $h$. Let $k$ be this bound. To decode, consider the first $k + 1$ bits on the hint tape and truncate the string $10^*$ from its right side. Namely, scan the $k + 1$ bits from right to left and truncate all zeros found until encountering a 1. Truncate this 1 too, and you are left with the decoded hint string.

[10] Here and throughout the paper we adopt the standard convention by which each oracle query is answered by a single bit.

The *knowledge complexity in the strict oracle sense*, denoted $\mathbf{kc}_{\text{oracle}}^{\text{strict}}$ (also $\mathbf{kc}_{\text{oracle}}^{1}$), and the class $\mathcal{KC}_{\text{oracle}}^{\text{strict}}$ (also $\mathcal{KC}_{\text{oracle}}^{1}$) are defined analogously to Definition 2.1.

Accessing the bits of the oracle sequentially, we can easily emulate a hint[11] and so we get:

**Proposition 2.3** *For every interactive proof* $\Pi$, $\mathbf{kc}_{\text{oracle}}^{\text{strict}}(\Pi) \leq \mathbf{kc}_{\text{hint}}(\Pi)$.

The second variant of oracle knowledge complexity allows the simulator to announce failure in half of its runs.

**Definition 2.4 (knowledge complexity - Oracle version):** *This definition is the same as the previous one except that condition (2) is substituted by :*

**2'.** *For each $x \in L$, machine $M_{V'}^{A}$ produces an output with probability at least $1/2$. Let $D_{V'}(x)$ denote the output distribution of $M_{V'}^{A}$ (condition that it produces an output at all). Then the ensembles $\{D_{V'}(x)\}_{x\in L}$ and $\{(P,V')(x)\}_{x\in L}$ are statistically close.*

The *knowledge complexity in the oracle sense*, denoted $\mathbf{kc}_{\text{oracle}}^{1/2}$ (also $\mathbf{kc}_{\text{oracle}}$) and the class $\mathcal{KC}_{\text{oracle}}^{1/2}$ (also $\mathcal{KC}_{\text{oracle}}$) are defined similarly to the previous definitions.

Clearly,

**Proposition 2.5** *For every interactive proof* $\Pi$, $\mathbf{kc}_{\text{oracle}}^{1/2}(\Pi) \leq \mathbf{kc}_{\text{oracle}}^{\text{strict}}(\Pi)$.

The constant $1/2$ used as a lower bound on the probability that the simulator produces an output, can be substituted by any constant $0 < \delta \leq 1$ to get $\mathbf{kc}_{\text{oracle}}^{\delta}(\Pi)$. The relation of this definition to the original one is given in the following two propositions:

**Proposition 2.6** *For every interactive proof* $\Pi$, *and every* $0 < \epsilon < \frac{1}{2}$,
$\mathbf{kc}_{\text{oracle}}^{1/2}(\Pi) \leq \mathbf{kc}_{\text{oracle}}^{\epsilon}(\Pi) + \lceil \log\left(\frac{1}{\epsilon}\right) \rceil$.

**Proof:** Given a simulator that produces an output with probability $\epsilon$, we build a new simulator that picks $2^{\lceil \log(\frac{1}{\epsilon}) \rceil} - 1$ random tapes for the original simulator and sends them to the oracle. The oracle specifies the first random tape on which the original simulator produces a conversation, or zero if no such tape exists. In the first case, the new simulator runs the original simulator on the selected tape, and refers its queries to the oracle. The probability that no output is produced by the new simulator is $(1-\epsilon)^{2^{\lceil \log(\frac{1}{\epsilon}) \rceil} - 1} < \frac{1}{2}$. $\Box$

**Proposition 2.7** *For every interactive proof* $\Pi$, *and every* $0 < \epsilon < \frac{1}{2}$,
$\mathbf{kc}_{\text{oracle}}^{1-\epsilon}(\Pi) \leq \mathbf{kc}_{\text{oracle}}^{1/2}(\Pi) + \lceil \log \lceil 1 + \log(\frac{1}{\epsilon}) \rceil \rceil$.

**Proof:** Similar to the proof of Proposition 2.6. This time the list is of length $\lceil \log(\frac{1}{\epsilon}) \rceil$, the failure probability is $(\frac{1}{2})^{\lceil \log(\frac{1}{\epsilon}) \rceil} \leq \epsilon$, and a pointer to an element in this list requires $\lceil \log(1 + \lceil \log(\frac{1}{\epsilon}) \rceil) \rceil$ bits. $\Box$

The third variant of the oracle knowledge complexity allows no failure, but instead allows the simulator more flexibility in the number of queries. This is done by considering the expected number of queries rather than the worst case number.

[11] Here we capitalize on the convention that the machine in Definition 2.1 reads the hint from an infinite tape and decides by itself when to stop reading.

**Definition 2.8 (knowledge complexity - Average Oracle version):** *This definition is the same as Definition 2.2, except that condition (1) is replaced by:*

**1'.** *On input $x$, the* average *number of queries that machine $M_{V'}$ makes to oracle $A$ is at most $k(|x|)$. (Here, the average is taken over the coin tosses of the machine $M_{V^*}$.)*

The *average knowledge complexity in the oracle sense*, denoted $\overline{\mathbf{kc}}_{\text{oracle}}$, and the class $\overline{\mathcal{KC}}_{\text{oracle}}$ are defined analogously to Definition 2.1.

### 2.3 The fraction measure

In the last definition the simulator is given no explicit help. Instead, only a $\frac{1}{2^k}$ fraction of its output distribution is being considered.

**Definition 2.9 (knowledge complexity - Fraction version):** *Let $\rho$: $\mathsf{N} \to (0,1]$. We say that an interactive proof $(P,V)$ for a language $L$ can be simulated at density $\rho(|x|)$ if for every probabilistic polynomial time verifier $V'$ there exists a probabilistic polynomial-time machine $M_{V'}$ with the following "good subspace" property. For any $x \in L$ there is a subset $S_x$ of $M_{V'}$'s possible random tapes such that:*

1. *The set $S_x$ contains at least a $\rho(|x|)$ fraction of all possible coin tosses of $M(x)$.*
2. *Let $D_{V'}(x)$ denote the output distribution of $M_{V'}^A$ conditioned on the event that $M_{V'}(x)$'s coins fall in $S_x$. Then the ensembles $\{D_{V'}(x)\}_{x\in L}$ and $\{(P,V')(x)\}_{x\in L}$ are statistically close.*

*The* knowledge complexity in the fraction sense*, denoted $\mathbf{kc}_{\text{fraction}}$, assigns the interactive proof $\Pi$ a function $k_\Pi$ so that $\Pi$ can be simulated at density $2^{-k_\Pi}$.* The class *$\mathcal{KC}_{\text{fraction}}(k(|x|))$ is defined analogously to the previous definitions.*

## 3 Examples and Remarks

We call the reader's attention to an interesting usage of knowledge complexity (§3.3), to a discussion regarding the effect of sequential repetition (§3.4), and to a variant of the definitional treatment in which one considers only honest verifiers (§3.5).

### 3.1 Focusing on the Statistical version

The presentation in this paper focuses on the *statistical* knowledge-complexity measures. An analogous treatment of the perfect and computational knowledge-complexity measures (for protocols) can be easily derived, except that we do not know whether the *perfect* knowledge-complexity analogue of Proposition 4.7 holds.

As per the knowledge-complexity hierarchies of languages, under some reasonable assumptions, the hierarchy of *computational* knowledge-complexity is quite dull. That is, assuming the existence of secure bit commitment scheme (i.e., the existence of one-way functions), all languages having interactive proofs have interactive proofs of *computational* knowledge-complexity zero [8, 23].

## 3.2 Analyzing the examples of the introduction

Alice's behavior in Examples 1 and 2 is zero-knowledge and thus of knowledge-complexity zero under all our definitions. In general, it is easy to see that zero-knowledge coincides with knowledge-complexity zero under all our definitions. As per Example 3, we can bound the knowledge-complexity (under each definition) by the length of the prime factorization (save the last factor which is easy to compute from the product and the other factors). We see no way to separate the various measures using this specific example.

Examples 4 and 5 are more interesting. Starting with Example 4, we observe that the discussion in Subsection 1.1 implies that the knowledge-complexity by the fraction measure is at most 1. The same holds with respect to the oracle measure (cf., Proposition 4.3). An obvious bound with respect to the hint measure is the length of the smallest element in $QNR_N^+$ (which under ERH has length $O(\log |N|)$ [25]), and it is not clear if one can provide a better bound.

As per Example 5, Alice sends a single bit which can be easily simulated by one oracle query (or by a good subspace of density 1/2). Thus, the knowledge-complexity of Alice's message under both the oracle and fraction measures is bounded by 1. In general, both the oracle and fraction measures are bounded above by the communication complexity in the Alice-to-Bob direction. Again, we don't know if such a bound can be obtained with respect to the hint measure. In this case the best bound we can offer is the length of the prime factorization of $N$ (again, save the last factor).

## 3.3 Another example: Parallel repetitions of some protocols

Consider the *basic* zero-knowledge interactive proof system (of soundness error 1/2) for Graph Isomorphism [13]: On input $x = (G_1, G_2)$, the prover generates a random isomorphic copy, denoted $H$, of $G_1$ and sends it to the verifier. The verifier uniformly selects $\sigma \in \{1, 2\}$ and the prover replies with the isomorphism between $H$ and $G_\sigma$.

**Example 7.** To obtain a zero-knowledge interactive proof system (with negligible error) for Graph Isomorphism, the basic protocol is repeated *sequentially* for $t(|x|) = \omega(\log |x|)$ times. In contrast, consider the protocol, denoted $\Pi_{\mathrm{GI}}^t$, resulting from $t(|x|)$ *parallel* repetition of the basic protocol. Indeed, $\Pi_{\mathrm{GI}}^t$ is also an interactive proof system (with negligible error) for Graph Isomorphism. However, unless Graph Isomorphism is in $\mathcal{BPP}$, protocol $\Pi_{\mathrm{GI}}^t$ is unlikely to be zero-knowledge [12]. Still, we can bound the knowledge complexity of $\Pi_{\mathrm{GI}}^t$ by $t$.

**Proposition 3.1** $\mathbf{kc}_{\mathrm{fraction}}(\Pi_{\mathrm{GI}}^t) \leq t(\cdot)$.

**Proof:** Use the obvious simulator (a la [13]): Uniformly select $\sigma_1, ..., \sigma_t \in \{1, 2\}$ and generate graphs $H_1, ..., H_t$ so that $H_i$ is a random isomorphic copy of $G_{\sigma_i}$. Send the sequence of graphs to the verifier. With probability $2^{-t}$ it will reply with the sequence $\sigma_1, ..., \sigma_t$, in which case we've produced a good simulation.[12] Otherwise, we output a "failure" symbol. $\square$

**Comment:** If $t = O(\log |x|)$ then we may invoke the simulator again until we get a good simulation, but this is not feasible for $t = \omega(\log |x|)$.

The above discussion applies to many other protocols. In particular, we mention M. Blum's zero-knowledge interactive proof system (of soundness error 1/2) for Hamiltonicity [11, Chap. 6, Exer. 15]. Thus, assuming the existence of one-way functions (as in [13]), there are *constant-round* interactive proofs of super-logarithmic computational knowledge complexity for any language in

[12] As in [13], any inappropriate response is interpreted as a canonical response, say the sequence $1, ..., 1$.

NP. Furthermore, these protocols are in the public–coin (Arthur–Merlin) model of Babai [5], and can be proven to have bounded knowledge complexity by using a black-box simulator. These protocols cannot be proven to be in zero-knowledge using a black-box simulator, unless $\mathcal{NP} \subseteq \mathcal{BPP}$ (cf., [12]).

### 3.4 The effect of sequential repetitions

Recall that zero-knowledge is preserved under sequential repetitions, provided that the definition is augmented to allow for auxiliary inputs: See [12] for demonstration of the necessity of an augmentation, and [14] for a definition of auxiliary-input zero-knowledge and a proof that it is preserved under sequential repetitions. It should thus come at no surprise that we start by augmenting our definitions so to handle auxiliary-inputs. Loosely speaking, the auxiliary input account for a-prior information that the verifier may have before entering the protocol. When defining zero-knowledge protocols, we wish that the verifier gains nothing from the interaction also in case it had such a-prior information. In the actual definitions, the simulator is given the same auxiliary input and is required to simulate the interaction relative to the common input and this auxiliary input. We require the same from the simulators in our definitions. For example, extending Definition 2.2, we present a definition of auxiliary-input knowledge-complexity in the strict oracle sense:

**Definition 3.2 (knowledge complexity with auxiliary input – strict oracle version):** *An interactive proof* $\Pi = (P, V)$ *for a language* $L$ can be strongly simulated (in the strict sense) with $k$ oracle queries*, if for every probabilistic polynomial-time verifier* $V'$ *there exists a probabilistic polynomial time oracle machine* $M_{V'}$ *and an oracle* $A$ *such that:*

1. *On input* $x \in L$ *and auxiliary input* $z$*, machine* $M_{V'}$ *queries the oracle* $A$ *at most* $k(|x|)$ *times.*
2. *The ensembles* $\{M_{V'}^{A}(x, z)\}_{x \in L, z}$ *and* $\{(P, V(z))(x)\}_{x \in L, z}$ *are statistically close.*

We stress that when defining knowledge complexity with auxiliary input *in the hint sense*, the hint remains a function of the common input (and is thus independent of the auxiliary-input).[13] This follows the motivation behind the hint measure, intended to capture a (predetermined) function of the input which may be leaked by the protocol.

**The effect of sequential repetition on the hint measure:** Knowledge complexity in the hint sense is preserved under sequential repetitions: For every interactive proof $\Pi = (P, V)$, and every polynomial-time computable function $t : \mathbb{N} \mapsto \mathbb{N}$ which is bounded by a polynomial, we let $\Pi^t$ denote the interactive proof system in which on input $x$ the system $\Pi$ is repeated $t(|x|)$ times and the verifier accepts iff the $V$ accepts in all $t(|x|)$ runs.

**Proposition 3.3** *For every interactive proof* $\Pi = (P, V)$*, if* $\Pi$ *can be strongly simulated using a hint of length* $k : \mathbb{N} \mapsto \mathbb{N}$ *then so can* $\Pi^t$*, for every function* $t : \mathbb{N} \mapsto \mathbb{N}$ *as above.*

**Proof outline:** The basic idea is to use the simulators guaranteed for $\Pi$ in order to construct simulators for $\Pi^t$. This is easy when the possibly cheating verifier (in $\Pi^t$) "respects" the structure of $\Pi^t$ (i.e., its actions in the $i^{\text{th}}$ run of $\Pi$ are independent of previous runs). However, this may not be true in general and overcoming the difficulties is done by adapting the ideas in [14]. Suppose

[13] Alternatively, one may adopt a more liberal measure in which the hint may be a function of both the common input and the auxiliary input. We do not know whether Proposition 3.3 holds under this alternative definition. Certainly, the alternative definition is at most additive (as in Proposition 3.4).

that $\Pi(x)$ can be strongly simulated on input $x$ when given the hint $h(x)$. Given a cheating verifier $V'$ for $\Pi^t$, we convert it into $t$ interactive machines, $V'_1, ..., V'_t$. The $i^{\text{th}}$ machine, $V'_i$, handles the interaction with $P$ during the $i^{\text{th}}$ run. Given an auxiliary input, which will be set as the view of the $(i-1)^{\text{st}}$ machine, machine $V'_i$ acts in the $i^{\text{th}}$ run as $V'$ would. Clearly, the view of $V'_t$ is identical to the final view of $V'$. To simulate $V'$, we use the simulators guaranteed for the $V'_i$'s. Note that all $V'_i$'s (except maybe the first) are actually identical and so these simulators are actually identical. The desired simulator (for $V'$) is obtained by applying this simulator $t$ times, feeding the output of the $(i-1)^{\text{st}}$ run into the $i^{\text{th}}$ run. The crucial observation is that all these runs of the simulator use the same hint, which is a function of the common input, and so the simulator for $V'$ just needs the same hint. □

**The effect of sequential repetition on the other measures:** In contrast to the above, the other measures of knowledge-complexity are not preserved under sequential repetitions – see Proposition 5.7. However, in all measures, knowledge-complexity is (essentially) at most additive. That is, let $\Pi = (P, V)$, $t : \mathsf{N} \mapsto \mathsf{N}$ and $\Pi^t$ be as above, then the knowledge complexity of $\Pi^t$ is (essentially) at most $t$ times the knowledge complexity of $\Pi$:

**Proposition 3.4** *For every interactive proof* $\Pi = (P, V)$ *and every polynomially bounded* $k, t : \mathsf{N} \mapsto \mathsf{N}$,

1. (strict oracle sense): *If* $\Pi$ *can be strongly simulated in a strict sense with* $k$ *oracle queries then* $\Pi^t$ *can be strongly simulated in a strict sense with* $tk$ *oracle queries.*
2. (average oracle sense): *If* $\Pi$ *can be strongly simulated with* $k$ *oracle queries on the average then* $\Pi^t$ *can be strongly simulated with* $tk$ *oracle queries on the average.*
3. (fraction sense): *If* $\Pi$ *can be strongly simulated at density* $2^{-k}$ *then* $\Pi^t$ *can be strongly simulated at density* $2^{-tk}$.
4. (oracle sense): *If* $\Pi$ *can be strongly simulated with* $k$ *oracle queries then* $\Pi^t$ *can be strongly simulated with* $t \cdot (k+1) + c$ *oracle queries, where* $c = 0$ *if* $t(n) = O(\log n)$ *and* $c = 4$ *otherwise.*

**Proof outline:** We merely follow the outline of the proof of Proposition 3.3, with the exception that here the same "help" can not be used in all runs. Thus, we need $k$ bits of "help" per each round. The argument varies slightly depending on the measure in use. In Item 1 (resp., Item 2) we make $k$ oracle queries (resp., on the average) during the simulation of each run, and thus we have a total of $tk$ queries (resp., on the average). In Item 3 we merely note that the good subspace is the Cartesian product of the good subspaces associated with individual runs. Item 4 is slightly subtle: Following the above argument we obtain a simulator which makes $tk$ queries and produces output with probability at least $2^{-t}$. We need to convert this simulator into one which produces output with probability at least $1/2$. Towards this end we use the techniques of the proof of Proposition 4.3. □

### 3.5 Knowledge complexity with respect to the honest verifier

It is possible to define knowledge complexity measures and knowledge complexity classes with respect to any fixed verifier, i.e., not requiring the existence of a simulation for all polynomial time verifiers but only for a specific one. An important and natural case is when we only require that there exists a simulation of the honest verifier (i.e., the verifier defined in the protocol). Intuitively, stating a knowledge complexity bound $k(\cdot)$ only for the honest verifier means that if the verifier follows his part in the protocol then he gains at most $k(|x|)$ bits of knowledge. However, if the verifier deviates from the protocol, then nothing is guaranteed and he may gain more than $k(|x|)$ bits of knowledge.

All the above definitions of knowledge complexity can be stated for this special case of the honest verifier simply by replacing the requirement that there exists a simulator for any possible verifier $V'$ with the requirement that there exists a simulator only for the honest verifier. For example, the definition of the oracle measure of knowledge complexity for the honest verifier (the analogue of Definition 2.4) is as follows.

**Definition 3.5 (knowledge complexity - Oracle version for honest verifier):** *We say that an interactive proof* $\Pi = (P, V)$ *for a language* $L$ has an honest verifier simulation with $k$ oracle queries, *if there exists a probabilistic polynomial time oracle machine* $M_V$ *and an oracle* $A$ *such that:*

1. *On input* $x \in L$, *machine* $M_V$ *queries the oracle* $A$ *at most* $k(|x|)$ *times.*
2. *For each* $x \in L$, *machine* $M_V^A$ *produces an output with probability at least* $1/2$. *Let* $D(x)$ *denote the output distribution of* $M_V^A$ *(condition that it produces an output at all). Then the ensembles* $\{D(x)\}_{x \in L}$ *and* $\{(P, V)(x)\}_{x \in L}$ *are statistically close.*

The *honest-verifier oracle knowledge-complexity measure* $\mathrm{hvkc}_{\mathrm{oracle}}$ and the class $\mathcal{HVKC}_{\mathrm{oracle}}$ are defined analogously to Definition 2.1.

We stress that all the results in this paper hold also with respect to the appropriate definitions for honest verifier. This holds since all our results are obtained by making transformations on the simulator, and never by modifying the protocol.

### 3.6 Remarks on the knowledge complexity function

Throughout the paper we consider only knowledge complexity functions which are polynomially bounded. This follows from the fact that we consider only simulators that run in polynomial time. Thus, the simulator can use only polynomially many oracle bits or hint bits. In the fraction measure, the good subspace $S_x$ must contain at least one string out of the (at most) exponentially many possible coin tosses, and thus (minus) the logarithm of the density of the good subspace is polynomially bounded.

Some of our results require that the knowledge complexity function be polynomial-time computable (see Definition 2.0).

## 4 Inclusion results

For every interactive proof $\Pi$, $\mathbf{kc}^{1/2}_{\mathrm{oracle}}(\Pi) \leq \mathbf{kc}^{\mathrm{strict}}_{\mathrm{oracle}}(\Pi) \leq \mathbf{kc}_{\mathrm{hint}}(\Pi)$ (see Propositions 2.3 and 2.5). In the following subsections we prove further inclusion relations between these definitions. We believe that the most interesting result (and proof), in this section, is that of Proposition 4.3.

### 4.1 The oracle and the fraction versions are equal up to a constant

In this subsection, we prove equivalence up to an additive constant of the oracle measure and the fraction measure of knowledge complexity. That is,

**Theorem 4.1** (closeness of the oracle and fraction measures): *For any interactive proof* $\Pi$*,*

$$\mathbf{kc}_{\text{fraction}}(\Pi) - 1 \le \mathbf{kc}^{1/2}_{\text{oracle}}(\Pi) \le \mathbf{kc}_{\text{fraction}}(\Pi) + 4$$

*Furthermore, for* $\mathbf{kc}^{1/2}_{\text{oracle}}(\Pi) = O(\log(\cdot))$

$$\mathbf{kc}_{\text{fraction}}(\Pi) - 1 \le \mathbf{kc}^{1/2}_{\text{oracle}}(\Pi) \le \mathbf{kc}_{\text{fraction}}(\Pi)$$

Theorem 4.1 follows from Propositions 4.2 and 4.3 below.

**Proposition 4.2** *For any interactive proof* $\Pi$*,* $\mathbf{kc}_{\text{fraction}}(\Pi) \le \mathbf{kc}^{1/2}_{\text{oracle}}(\Pi) + 1$*.*

**Proof:** Let $k = \mathbf{kc}^{1/2}_{\text{oracle}}(\Pi)$. Suppose there is a simulator $M$ that makes at most $k$ queries to the oracle and produces an output with probability at least $\frac{1}{2}$. We construct a new simulator $M'$ which does not make oracle queries but has a good sub-space of density at least $2^{-(k+1)}$. The simulator $M'$ randomly selects a random tape to the original simulator $M$ and guesses (at random) $k$ bits representing the oracle answers. The new simulator $M'$ runs $M$ on this random tape, and uses the $k$ random bits to "reply to the queries" of $M$ instead of the oracle. With probability at least $2^{-k}$ the original simulator $M$ gets the correct oracle answers, and conditioned on this event, $M$ produces an output with probability at least $1/2$. Thus, with probability at least $2^{-(k+1)}$, the original simulator $M$ both gets the correct oracle answers and does yield an output. In this case, $M'$ produces exactly the same output distribution as the original simulator. $\square$

**Proposition 4.3** *For any interactive proof* $\Pi$ *with polynomial time computable knowledge complexity in the fraction sense,* $\mathbf{kc}^{1/2}_{\text{oracle}}(\Pi) \le \mathbf{kc}_{\text{fraction}}(\Pi) + 4$*.*
(For perfect knowledge-complexity we have $\mathbf{kc}^{1/2}_{\text{oracle}}(\Pi) \le \mathbf{kc}_{\text{fraction}}(\Pi) + 5$.)
*Furthermore, if* $\mathbf{kc}_{\text{fraction}}(\Pi) = O(\log(\cdot))$ *then* $\mathbf{kc}^{1/2}_{\text{oracle}}(\Pi) \le \mathbf{kc}_{\text{fraction}}(\Pi)$*.*

**Proof:** Below (as well as in some other proofs), we transform a simulator guaranteed by one definition into a simulator satisfying a second definition. Our transformation assumes the "knowledge" of the value of the knowledge-complexity function on the specific input length. Thus, the constructed simulator starts by computing this value (hence the condition that the knowledge-complexity function be polynomial-time computable).

Let $k = \mathbf{kc}_{\text{fraction}}(\Pi)$. Our purpose is to pick uniformly at random $r \in S_x$ and run the original simulator on $r$. The problem is how to sample $S_x$. The naive solution of asking the oracle to pick an $r \in S_x$ uniformly, is not good, since $r$ might be much longer than $k(|x|)$. A better suggestion, that works for $k = O(\log|x|)$ follows. Pick a list of $2^k - 1$ random strings for the original machine, and ask the oracle to specify one of them that belongs to $S_x$. The oracle answers with the index of the first string which belongs to $S_x$, if such exists, and otherwise returns the all-zero string. The length of the list is polynomial in $|x|$ because $k = O(\log|x|)$. The probability of failing (not having any good string in the list) is $(1 - \frac{1}{2^k})^{2^k - 1} \le \frac{1}{2}$ (with equality holding for $k = 1$). Hence in this case (i.e., for $k = O(\log|x|)$), we have $\mathbf{kc}^{1/2}_{\text{oracle}}(\Pi) \le \mathbf{kc}_{\text{fraction}}(\Pi)$.

Dealing with $k = \frac{\log|x|}{o(1)}$ is somewhat more complicated. We would like to do the same as before, but the problem is that the list of length $2^k - 1$ is not polynomial (in $|x|$), and thus cannot be given explicitly. In the rest of this proof, we are going to show that by somewhat sacrificing the "full" randomness of the list, we can still follow a procedure similar to the above in polynomial time. Namely, we will use a "pseudorandom" list of $2^{k+2} - 1$ strings so that

**(1)** The list can be succinctly represented and the representation enables efficient retrieval of its elements (i.e., given the succinct representation and an index $i$ into the list we can efficiently find the $i^{\text{th}}$ string in the list);

**(2)** Uniformly selecting such a representation yields a sequence which is sufficiently random in the sense that with constant probability the list intersects $S_x$;

Combining items (1) and (2), we get an efficient procedure for sampling $S_x$ using $k+2$ oracle queries. The procedure consists of selecting at random a (succinct representation of a) list $L$ and sending it to the oracle which replies with the index of an element in $L$ which resides in $S_x$ (if such exist). A reasonable implementation of this procedure is likely to output each of the elements in $S_x$ with some probability (say between $O(1)/|S_x|$ and $1/O(|S_x|)$), but is unlikely to sample $S_x$ uniformly. Let us stress that it is important that the procedure uniformly samples $S_x$, the reason being that we will use the output of the procedure for running the original simulator which is guaranteed to produce the correct output only when being run with coins uniformly selected in the good set $S_x$. We thus need an alternative procedure, which using $k + O(1)$ oracle queries samples $S_x$ uniformly–

**(3)** There exists an oracle such that the following procedure samples uniformly in $S_x$. The procedure picks a random list $L$ and sends the representation of the list to the oracle. The oracle returns $k+2$ bits which either provide an index to an element in the list or indicates failure. In the first case the procedure outputs the string pointed to by the index and in the second case there is no output. We require that with constant probability the procedure yields output, and that in case an output is produced it is uniformly distributed in $S_x$.

A natural way for generating a random list satisfying Items (1) and (2) above is to use a sequence of pairwise independent random variables. Actually, our analysis becomes even easier if we use a sequence of 3-wise independent random variables. It will be most convenient to use the construction given in Alon et. al. [4] which works in the field $GF(2^m)$, since this field corresponds naturally to the set of $m$-bit strings. The construction uses $t < 2^m$ arbitrary elements of the field, denoted $\alpha_1, \alpha_2, ..., \alpha_t$. A sequence of $t$ elements is represented by a triplet of field elements, denoted $(u, v, w)$, and the $i^{\text{th}}$ element in the sequence is $u + \alpha_i \cdot v + \alpha_i^2 \cdot w$ (where all operations are in the field). Clearly, this construction satisfies Item (1) above. It is also well-known that a uniformly selected triplet $(u, v, w)$ induces a $t$-long sequence which is 3-wise independent so that each element is uniformly distributed in $GF(2^m)$.[14] For $t = 2^{k+2} - 1$, it follows (by Chebyshev's Inequality) that any set $S \subseteq \{0,1\}^n$ of cardinality at least $2^{m-k}$ is hit with constant probability. Thus, Item (2) is satisfied as well. Before proceeding, let us remark that we may assume, without loss of generality, that $2^{k+2} - 1 < 2^m$ (as otherwise we can use the trivial procedure of asking the oracle for a (random) element in $S_x$).

We are left with the task of satisfying Item (3). A natural implementation of the above sampling procedure is to ask the oracle for a random[15] element in the intersection of the list and the subset $S_x$. The intersection is expected to have size $(2^{k+2} - 1) \cdot \frac{|S_x|}{2^m} \approx 4$ (assuming[16], for simplicity, that $|S_x| = 2^{m-k}$). If the intersection of a uniformly chosen sequence and $S_x$ *always* had size equal to its expectation then we would have been done. However, this is unlikely to be the case. We can only

[14]This can be seen by considering the equalities which relate to any three elements in the sequence (i.e., the elements in positions $i$, $j$ and $k$). These equations form a non-singular transformation of the triplet $(u, v, w)$ to the values of these three elements. (Thus, all possible $2^{3m}$ outcomes are possible and appear with the same probability.)

[15]The oracle can be made to take "random" moves by supplying it with coin tosses (by appending them to the query).

[16]Recall that we are only guaranteed that $|S_x| \geq 2^{m-k}$).

be guaranteed that with high probability the size of the intersection is *close* to its expectation and sometimes even this does not hold (i.e., with small probability the size of the intersection is far from the expectation). Our solution uses two natural ideas. First, we ignore the unlikely cases in which the intersection is far from the expectation. Secondly, we skew the probabilities to compensate for the inaccuracy. To be more specific, let $I$ denote the intersection (of the list and $S_x$) and let $e$ denote the expected size of the intersection. In case $|I| > 2e$ the oracle indicates failure. Otherwise, it indicates failure with probability $1 - \frac{|I|}{2e}$ and outputs each element in $I$ with probability exactly $\frac{1}{2e}$. This skewing is not sufficient, since some elements in $S_x$ may tend to appear, more often than others, in lists which have too large intersection with $S_x$. Yet, we can correct this problem by a more refined skewing. A detailed description follows.

Recall that we are given an (original) simulator which simulates the protocol with density $2^{-k(\cdot)}$. Fix an input $x$ to the protocol, define $k = k(|x|)$, and denote by $m = m(|x|)$ the length of the random tape used by the original simulator on input $x$. Set $\rho = |S_x|/2^m$ and $t = 2^{k+2} - 1$. We are going to build a new simulator which uses $k + O(1)$ oracle queries and runs the original simulator as a subroutine (see Step (4)). The new simulator will produce an output with constant probability and its output will be distributed identically to the distribution produced by the original simulator when its coins are taken from the "good sub-space". Thus, the new simulator is a good one (i.e., it simulates the original prover-verifier interaction).

**The new simulator (for the oracle definition of knowledge complexity):**

1. The simulator picks uniformly a triplet $(u, v, w)$ so that $u, v, w \in GF(2^m)$. This triplet is a succinct representation of the $t$-long sequence

$$L = L(u, v, w) \stackrel{\text{def}}{=} (u + \alpha_i v + \alpha_i^2 w)_{1 \le i \le t}$$

   The simulator sends $(u, v, w)$ to the oracle.

2. The oracle answers as follows:

   (a) If either $|L \cap S_x| = 0$ or $|L \cap S_x| > 2 \cdot (t \cdot \rho)$, then the oracle returns "failure" (i.e., $0^{k+2}$). Here and below $L \cap S_x$ is a multi-set and so

$$|L(u, v, w) \cap S_x| = |\{i : u + \alpha_i v + \alpha_i^2 w \in S_x\}|$$

   (b) Otherwise, the oracle picks at random $r \in L \cap S_x$. Actually, it picks uniformly an element in $\{i : u + \alpha_i v + \alpha_i^2 w \in S_x\}$.

   (c) With probability $P_{x,i,L}$ (to be specified below), the oracle returns the index $i$ chosen in Sub-step (b) above.

   (d) Otherwise (with probability $1 - P_{x,i,L}$), the oracle returns "failure".

3. If the oracle returns "failure", then the simulator stops with no output.

4. Otherwise, upon receiving answer $i$ $(1 \le i \le t)$, the simulator computes $r \leftarrow u + \alpha_i v + \alpha_i^2 w$, and runs the original simulator using $r$ as its random tape.

**Motivation:** Let us first assume, for simplicity, that $|S_x| = 2^{m+2}/t \approx 2^{m-k}$ and that the cardinality of $L \cap S_x$ always equals its expected value (i.e., $|L \cap S_x| = \frac{2^{m+2}}{t} \cdot \frac{t}{2^m} = 2^2$). This means that the oracle never answers "failure" in Step (2a). Setting $P_{x,r,L} = 1$, the oracle never answers "failure" in Step (2d) either. Since each element of the list is uniformly distributed, it follows that each $r \in S_x$ is selected with probability $\frac{2^2}{|S_x|} \cdot \frac{1}{2^2} = \frac{1}{|S_x|}$. However, this simple analysis does not suffice in general since the "uniform behavior" of the cardinality of $L \cap S_x$ cannot be guaranteed. Instead, an approximate behavior can be shown to occur.

**Analysis:** We call a list $L$ bad if either $|L \cap S_x| = 0$ or $|L \cap S_x| > 2 \cdot (t \cdot \rho)$. Otherwise, the list is called good. Thus, the oracle answers "failure" in Step (2a) if and only if it is presented with a bad list. For every $r \in S_x$ and every $i \leq t$, we denote by $L_{i,r}$ the set of lists in which $r$ appears in the $i^{\text{th}}$ position; namely

$$L_{i,r} \stackrel{\text{def}}{=} \{(u, v, w) : r = u + \alpha_i v + \alpha_i^2 w\}$$

Clearly, $|L_{i,r}| = 2^{-m} \cdot 2^{3m}$ (as each element in a random list is uniformly distributed). Now let $G_{i,r}$ denote the subset of good lists in $L_{i,r}$ (i.e., $L \in G_{i,r}$ iff $L$ is good and in $L_{i,r}$). Using the fact that lists are 3-wise independent, we show (see Lemma 4.4 below) that at least $\frac{1}{2}$ of the lists in $L_{i,r}$ are in $G_{i,r}$. We now list a few elementary facts regarding any fixed $r \in S_x$

1. The probability that $r$ is used in Step (4) of the simulator is the sum over all $i$'s of the probabilities that the oracle returns $i$ in Step (2c) and $r = u + \alpha_i v + \alpha_i^2 w$ holds.

2. The probability that the oracle returns $i$ and $r = u + \alpha_i v + \alpha_i^2 w$ holds equals $p_1 \cdot p_2 \cdot p_3$, where

$$\begin{aligned} p_1 &\stackrel{\text{def}}{=} \text{Prob}(L \in L_{i,r}) \\ p_2 &\stackrel{\text{def}}{=} \text{Prob}(L \in G_{i,r} | L \in L_{i,r}) \\ p_3 &\stackrel{\text{def}}{=} \sum_{L \in G_{i,r}} \frac{1}{|G_{i,r}|} \cdot \left( \frac{1}{|L \cap S_x|} \cdot P_{x,i,L} \right) \end{aligned}$$

   where the probabilities (in the first two definitions) are taken uniformly over all possible choices of $L$. Recall that $p_1 = 2^{-m}$ and that (we'll show) $p_2 > \frac{1}{2}$.

3. Setting $P_{x,i,L} \stackrel{\text{def}}{=} \frac{|L \cap S_x|}{2t\rho} \cdot q_{i,r}$, where $r$ is the $i^{\text{th}}$ element in $L$ (and $q_{i,r}$ will be determined next), we get $p_3 = \frac{q_{i,r}}{2t\rho}$. Note that $q_{i,r} \leq 1$ guarantees that $P_{x,i,L} \leq 1$ (since $|L \cap S_x| \leq 2t\rho$).

4. Knowing that $p_2 = \frac{|G_{i,r}|}{|L_{i,r}|} \geq \frac{1}{2}$, we set $q_{i,r} \stackrel{\text{def}}{=} \frac{1/2}{p_2}$ (which guarantees $q_{i,r} \leq 1$). We conclude that the probability that the oracle returns $i$ and $r = u + \alpha_i v + \alpha_i^2 w$ holds equals

$$2^{-m} \cdot p_2 \cdot \frac{q_{i,r}}{2t\rho} = 2^{-m} \cdot \frac{1/2}{2t\rho}$$

   Consequently, the probability that $r$ is used in Step (4) of the simulator equals

$$\begin{aligned} t \cdot \left( 2^{-m} \cdot \frac{1/2}{2t\rho} \right) &= \frac{1}{4} \cdot \frac{1}{2^m \rho} \\ &= \frac{1}{4} \cdot \frac{1}{|S_x|} \end{aligned}$$

   This means that with probability $1/4$ our simulator produces an output and that this output is produced by invoking the original simulator using a coin sequence uniformly chosen in $S_x$.

A few minor things require attention. Firstly, in the above description we have required the oracle to make random choices, an action which it cannot do. To overcome this problem, we simply let the simulator supply the oracle with extra coin-tosses which the oracle uses for implementing its random choices. (This works well if it suffices to approximate the probabilities in question, as is the case when treating *statistical* knowledge-complexity; we'll remark about handling *perfect* knowledge complexity at a later stage). Another minor difficulty is that the simulator described above outputs conversations with probability $\frac{1}{4}$ (and outputs nothing otherwise). Hence, we only proved $\mathbf{kc}^{1/4}_{\text{oracle}}(\Pi) \leq kc_{Fraction}(\Pi)+2$. Yet, using Proposition 2.6,[17] the current proposition follows. We now turn to prove the following

**Lemma 4.4** *For every $i \leq t$ and every $r \in S_x$,*

$$\text{Prob}(L \in G_{i,r} | L \in L_{i,r}) \geq \frac{5}{9} - 2^{-k} > \frac{1}{2}$$

**Proof:** Using the hypothesis that the sequence is 3-wise independent, it follows that fixing the $i^{\text{th}}$ element to be $r$ leaves the rest of the sequence pairwise independent (and uniformly distributed over $\{0,1\}^m$). Assume, without loss of generality, that $i = t$ and let $s = t - 1$. Define random variables $\zeta_j$ representing whether the $j^{\text{th}}$ element of the ($s$-long) sequence hits $S_x$. Recall, $\rho = |S_x|/2^m$ and thus $\zeta_j = 1$ with probability $\rho$ and $\zeta_j = 0$ otherwise. Now, the event that we are interested in is rewritten as $\sum_{j=1}^{s} \zeta_j > 2t\rho - 1$, where the $-1$ is due to the fact that the $t^{\text{th}}$ element in the $t$-long sequence is assumed to be in $S_x$. Thus, using Chebyshev's Inequality we get

$$\begin{aligned}
\text{Prob}(L \notin G_{i,r} | L \in L_{i,r}) &= \text{Prob}(\sum_{j=1}^{s} \zeta_j > 2t\rho - 1) \\
&< \text{Prob}(|\sum_{j=1}^{s} \zeta_j - s\rho| > s\rho - 1) \\
&\leq \frac{s \cdot \rho(1-\rho)}{(s\rho - 1)^2}
\end{aligned}$$

Using $s\rho \geq (2^{k+2} - 2) \cdot 2^{-k} = 4 - 2^{-k+1}$, the above expression is bounded by $\frac{4}{9} + 2^{-k}$ and the lemma follows (since we may assume $k > 5$ or else $k = O(\log n)$). □

The proposition now follows (for *statistical* knowledge complexity). For the case of *perfect* knowledge complexity, we are left with one problem; namely, implementing the random choices required of the oracle with *exactly* the prescribed probability. To this end, we slightly modify the procedure as follows. First, we observe that the two random choices required of the oracle can be incorporated into one. Indeed, an alternative description of the oracle's choices is that given a good list $L$ (i.e., $|L \cap S_x| \leq 2t\rho$) which contains $r \in L \cap S_x$ as its $i^{\text{th}}$ element, the oracle answers $i$ with probability $\frac{1}{|L \cap S_x|} \cdot P_{x,i,L} = q_{i,r} \cdot \frac{1}{2t\rho}$. Replacing $2t\rho$ by the next power of 2 (i.e., $2^{\lceil \log_2 2t\rho \rceil}$) maintains the validity of the procedure (since the decrease in the probability of using $r \in S_x$ is equal for all such $r$'s) while possibly decreasing the probability of output by up to a factor of 2 (see Fact 4 in the list of elementary facts above). The advantage of this modification is that the oracle can easily handle probabilities which are of the form $q/2^l$ for integers $q$ and $l$. So given a good list $L$ (i.e., $|L \cap S_x| \leq 2^{\lceil \log_2 2t\rho \rceil}$), we can select each index $i$ corresponding to an $r \in L \cap S_x$ with probability $2^{-\lceil \log_2 2t\rho \rceil}$. We are left with the problem of implementing the additional "sieve" corresponding to probability $q_{i,r}$ which equals $\frac{1/2}{|G_{i,r}|/|L_{i,r}|} = \frac{|L_{i,r}|}{2|G_{i,r}|}$. This probability is not of the "admissible" form,

[17] We will ask the oracle to indicate which of three uniformly chosen random-tapes (for the $\frac{1}{4}$-simulator) is going to produce an output. This requires two additional queries and the probability that none of the three random-tapes produces an output is at most $(3/4)^3 < \frac{1}{2}$.

yet we observe that the very same probability needs to be implemented for every $L \in G_{i,r}$. So, instead, we may designate $\frac{|L_{i,r}|}{2}$ ($\leq |G_{i,r}|$) of the lists in $G_{i,r}$ and modify the oracle so that it returns the index $i$ (corresponding to $r$) only when presented with these lists. To summarize, the modified oracle procedure for the case of perfect knowledge complexity is as follows.

**(notation)** Let $L = L(u, v, w)$ be the list indicated in the query and $D \stackrel{\text{def}}{=} 2^{\lceil \log_2(2\cdot(t\cdot\rho))\rceil}$. For every $i$ and $r$, let $D_{i,r}$ be an arbitrary subset of $G_{i,r}$ having cardinality $\frac{|L_{i,r}|}{2}$.

**(a')** If $|L \cap S_x| > D$, then the oracle returns "failure".

**(b')** Otherwise, the oracle uniformly selects $j \in \{1, 2, ..., D\}$. If $j > |L \cap S_x|$ then the oracle returns "failures"; otherwise, $i$ is defined as the index in $L$ of the $j^{\text{th}}$ element in $L \cap S_x$ and $r$ is the element itself (i.e., $r = u + \alpha_i v + \alpha_i^2 w$).

**(c')** If $L \in D_{i,r}$ then the oracle returns the index $i$ chosen above.

**(d')** Otherwise (i.e., $L \in G_{i,r} - D_{i,r}$) the oracle returns "failure".

Using the modified oracle, our simulator produces an output with probability at least 1/8 (and the output distribution is identical to the output distribution of the original simulator when run on a random tape uniformly selected in $S_x$). Using Proposition 2.6,[18] the current proposition follows. □

**Remark 4.5** *Our original proof of Proposition 4.3 (cf., [16]) used a more complicated construction of a "somewhat random" list with no apparent advantage. Furthermore, the additive constant achieved there (i.e., 11) is worse.*

## 4.2 The oracle vs. average oracle measure

**Proposition 4.6** *For any interactive proof* $\Pi$, $\overline{\mathbf{kc}}_{\text{oracle}}(\Pi) \leq \mathbf{kc}^{1/2}_{\text{oracle}}(\Pi) + 2$.

**Proof:** Suppose we have a probabilistic polynomial time oracle machine (a simulator) that queries the oracle $k(|x|)$ times, outputs a conversation with probability at least $\frac{1}{2}$, and its output distribution is statistically close to $\Pi$. We construct a new simulator which chooses random coin-tosses for the original simulator and asks the oracle whether these coins are "good", i.e., whether on this random string the original simulator (querying the original oracle) outputs a conversation (rather then nothing). If the oracle says "yes", the original simulator is run on these coin-tosses, and its $k(|x|)$ queries are answered by the oracle. Otherwise the new simulator tries again. In order not to allow infinite runs, we let the new simulator make up to $p(|x|)$ tries, where $p$ is a polynomial bounding the length of the conversations in $\Pi$. If all tries fail, the new simulator asks the oracle for a conversation. The probability that the new simulator succeeds in each try is at least $\frac{1}{2}$, and therefore if the simulator were allowed to make infinite runs, we would have got that the expected number of additional queries made by the new simulator would have been at most 2. Note that forcing the simulator to stop after $p(|x|)$ tries, and then make additional $p(|x|)$ queries, does not increase the average number of additional queries. □

In light of the results in Section 5, it is not possible to bound the opposite direction (i.e., it is not even possible to prove that $\mathbf{kc}^{1/2}_{\text{oracle}}(\Pi) \leq \overline{\mathbf{kc}}_{\text{oracle}}(\Pi) + p(|x|)$ for some polynomial $p$ and all $\Pi$'s).

[18] This time we use 7 random-tapes (and $(7/8)^7 < 1/2$) and this requires three additional queries.

### 4.3 The oracle and the strict oracle versions are close

Clearly, as stated in Proposition 2.5, $\mathbf{kc}^{\text{strict}}_{\text{oracle}}(\Pi) \geq \mathbf{kc}^{1/2}_{\text{oracle}}(\Pi)$. Proposition 4.7 gives us a complementary relation.

**Proposition 4.7** *For any interactive proof* $\Pi$ *and any unbounded function* $g : N \to N$, $\mathbf{kc}^{\text{strict}}_{\text{oracle}}(\Pi) \leq \mathbf{kc}^{1/2}_{\text{oracle}}(\Pi) + \log(\log(|x|)) + g(|x|)$.

**Proof:** Use Proposition 2.7 with $\epsilon = \frac{1}{|x|^{f(|x|)}}$, where $f(|x|) = \min\{2^{g(|x|)}, |x|\}$. Note that $\epsilon$ is a negligible fraction and that Proposition 2.7 holds as long as $\epsilon > 2^{-poly(|x|)}$. We get a simulator which does not produce an output with probability at most $\epsilon$. We modify this simulator so that it always produces an output. (Note that since we are using the statistical version of knowledge complexity it doesn't matter what the output is in this case, which happens with a negligible probability). $\Box$

In light of the results in Section 5, the above result cannot be improved in general.

## 5 Separation results

In this section we provide separation results for the knowledge complexity of specific protocols. We stress that these results do not necessarily translate to a separation of languages according to their knowledge complexity (which would have implied a separation between $\mathcal{BPP}$ and $\mathcal{PSPACE}$). As mentioned in Subsection 3.6, we consider only knowledge complexity functions which are polynomially bounded. This fact is required in all the proofs of this section.

### 5.1 Strictness of the hierarchies

We first show a separation in each of the knowledge complexity hierarchies. Namely, we show that each of these hierarchies (of protocols) is strict.

**Theorem 5.1** (strictness of hierarchies): *Let* $k : \mathsf{N} \to \mathsf{N}$ *be a polynomial time computable function. Then there exists an interactive proof system* $(P, V)$ (for the language $\{0,1\}^*$) *satisfying:*

1. $(P, V)$ *has knowledge complexity* at most $k(|x|)$ *in the hint sense* (and also in all other senses considered in this paper).
2. *(a)* $(P, V)$ *has knowledge complexity* at least $k(|x|)$ *in the oracle sense, provided* $k(n) = O(\log n)$.
   *(b)* $(P, V)$ *has knowledge complexity* at least $k(|x|) - 1$ *in the oracle sense.*
   *(c)* $(P, V)$ *has knowledge complexity* at least $k(|x|)$ *in the* strict *oracle sense.*
3. *For any polynomial* $p$*, the interactive proof system* $(P, V)$ *has knowledge complexity* greater than $k(|x|) - \frac{1}{p(|x|)}$ *in the fraction sense.*
4. *For any polynomial* $p$*, the interactive proof system* $(P, V)$ *has knowledge complexity* greater than $k(|x|) - 2 - \frac{1}{p(|x|)}$ *in the average oracle sense.*

A lower bounds of $k(|x|)$ hold also for the hint measure, since $\mathbf{kc}^{\text{strict}}_{\text{oracle}}(\Pi) \leq \mathbf{kc}_{\text{hint}}(\Pi)$ for every interactive proof $\Pi$. We believe that Part (2) can be improved so that it holds that $(P, V)$ has knowledge complexity at least $k(|x|)$ in the oracle sense, for any $k : \mathsf{N} \to \mathsf{N}$.

**Overview of the proof:** We consider a generic (artificial) interactive proof (for the language $\{0,1\}^*$) in which, on input $x$, the prover sends to the verifier a single string, denoted $K(x)$, to be specified later. The function $K : \{0,1\}^* \to \{0,1\}^*$ will satisfy $|K(x)| = k(|x|)$, for every $x \in \{0,1\}^*$, and will be chosen so to fail all simulators which violate the lower bounds of Parts (2)–(4). Part (1) follows easily by using a trivial hint-version simulator which merely outputs its hint.

To prove Parts (2)–(4) we use the diagonalization technique. We will show that it is possible to choose the function $K(\cdot)$ such that all Turing machines fail to simulate this protocol unless they receive the help that is stated in the theorem (i.e., enough hint bits, enough oracle bits, or the possibility to output a small enough "good" subspace $S_x$). Specifically, each part of the theorem asserts a lower bound on the help needed to simulate the protocol properly. We are going to select the function $K(\cdot)$ so that all lower bounds hold. For each lower bound and each possible Turing machine, $M$, which may serve as a simulator, we select a different input $x$ and set the function $K$ at $x$ such that the machine $M$ on input $x$ does not simulate the transcript $K(x)$ unless it gets enough help as stated in the lower bound. Actually, this description, adopted in the rest of the proof, suffices only for foiling *perfect* simulations (and thus establishing *perfect* knowledge-complexity lower bounds). To foil *statistically close* simulations (and establish *statistical* knowledge-complexity lower bounds), we need to select a (different) infinite sequence of inputs per each possible machine (since any two finite sequences of random variables may be said to be statistically close).

For simplicity, we discuss each lower bound separately, and foil all machines that do not get enough help as stated in the specific lower bound. However, one should keep in mind that we are building a single function $K$ which satisfies all the lower bounds simultaneously.

Our proof proceeds as follows. We first prove Part (3) of the theorem, next Part (2), and conclude the proof of the theorem by proving Part (4).

**Proof of Part 3:** Here we show that for an appropriate choice of a function $K$, the above interactive proof has knowledge complexity greater than $t(|x|) \stackrel{\text{def}}{=} k(|x|) - \frac{1}{p(|x|)}$ in the fraction sense for any given polynomial $p$. The function $K$ is constructed using the diagonalization technique. Consider an enumeration of all probabilistic polynomial time machines. For each machine $M$, select a different input $x$, and consider the output distribution $M(x)$. Machine $M$ simulates the interactive proof with density $2^{-t}$, if it outputs the string $K(x)$ on input $x$ with probability at least $\frac{1}{2^{t(|x|)}} \cdot (1-\epsilon)$, where $\epsilon$ is a negligible fraction. Note that $\frac{1}{2^{t(|x|)}} \cdot (1-\epsilon) > \frac{1}{2^{k(|x|)}}$ since $\epsilon$ is smaller than any polynomial fraction. By a pigeon hole argument, there is a string $y$ of length $k(|x|)$, that appears in $M(x)$ with probability at most $\frac{1}{2^{k(|x|)}}$, and setting $K(x) = y$ foils the simulation of $M$ on $x$.[19] Part (3) follows. □

**Proof of Part 2:** Here we show how to foil all oracle machines that never ask more than $k(|x|)-1$ queries (and must produce an output with probability at least $\frac{1}{2}$). We consider an enumeration of all probabilistic polynomial time *oracle* machines, and select a different input $x$ for each machine $M$. As a mental experiment, consider the output distribution of machine $M$ on input $x$ and access to a *random* oracle $\alpha$. Let us denote by $p$ the probability that on input $x$ and access to a random oracle, machine $M$ produces output. Again, by a pigeon hole argument, there is a string $y$ of length $k(|x|)$, that appears in this distribution with probability less or equal to $2^{-k(|x|)} \cdot p$. We show that for any *specific* oracle $\alpha$, the probability that this string $y$ appears in the output distribution of machine $M$ on input $x$ and access to the oracle $\alpha$ is $\frac{p}{2}$ at the most. Let $k = k(|x|)$ and let $m = m(|x|)$ be

[19]Note that we did not use the fact that $M$ is a polynomial time machine. It is indeed true that we can set the function $K$ such that this protocol cannot be simulated with density $2^{-t(|x|)}$ even by a (non-restricted) Turing machine. A similar observation is valid for all the lower bounds in this theorem.

a bound on the running time of machine $M$ on input $x$ (and therefore, a bound on the number of its coin-tosses). Denote by $r \in \{0,1\}^m$ the coin-tosses of machine $M$, and by $A(x,r,\alpha)$ the $k-1$ answers that oracle $\alpha$ gives machine $M$ on input $x$ and random tape $r$. Note that $x, r$ and $A(x,r,\alpha)$ completely determine the behavior of machine $M$. Also note that the random oracle $\alpha$ is chosen independently of $r$. Let us denote the output of machine $M$ on input $x$, random string $r$ and access to oracle $\alpha$ by $M_r^\alpha(x)$. The probability that the string $y$ appears in the output distribution of machine $M$ with access to a specific oracle $\alpha_0$ is

$$\begin{aligned} \mathrm{Prob}_r(M_r^{\alpha_0}(x)=y) &= \mathrm{Prob}_{r,\alpha}(M_r^\alpha(x)=y|A(x,r,\alpha)=A(x,r,\alpha_0)) \\ &\leq \frac{\mathrm{Prob}_{r,\alpha}(M_r^\alpha(x)=y)}{\mathrm{Prob}_{r,\alpha}(A(x,r,\alpha)=A(x,r,\alpha_0))} \end{aligned}$$

The numerator is at most $2^{-k}\cdot p$ (by the choice of $y$). We now show that the denominator is equal to $\frac{1}{2^{k-1}}$.

$$d \stackrel{\mathrm{def}}{=} \mathrm{Prob}_{r,\alpha}(A(x,r,\alpha)=A(x,r,\alpha_0)) = \frac{1}{2^m}\sum_{r\in\{0,1\}^m} \mathrm{Prob}_\alpha(A(x,r,\alpha)=A(x,r,\alpha_0))$$

Since $r$ (and $x$) are fixed inside the summation, a random choice of $k-1$ oracle answers agrees with the answers of the specific oracle $\alpha_0$ with probability $\frac{1}{2^{k-1}}$ and we get:

$$d = \frac{1}{2^m}\sum_{r\in\{0,1\}^m}\frac{1}{2^{k-1}} = \frac{1}{2^{k-1}}$$

Thus, for any specific oracle $\alpha_0$, the string $y$ appears in the output distribution of $M^{\alpha_0}(x)$ with probability at most $\frac{p}{2}$. Selecting $K(x)=y$, causes $M$ to produce $K(x)$ with probability at most $\frac{p}{2}$ (no matter which oracle is being used). But what does this mean? We consider two cases:

**Case 1:** $p > 1-2^{-(k+1)}$. We claim that in this case, for any specific oracle, machine $M$ produces (some) output with probability at least $\frac{3}{4}$. This holds since otherwise $p < 2^{-(k-1)}\cdot\frac{3}{4}+(1-2^{-(k-1)})\cdot 1 = 1-2^{-(k+1)}$. Thus, for any $\alpha$, the probability that $M^\alpha(x)=K(x)$ conditioned on $M^\alpha(x)$ producing output is at most $\frac{p/2}{3/4}\leq\frac{2}{3}$, which means that $M$ fails to simulate $(P,V)(x)$ properly.

**Case 2:** $p \leq 1-2^{-(k+1)}$. In this case, for any $\alpha$, the probability that $M^\alpha(x)=K(x)$ is at most $\frac{p}{2}\leq\frac{1}{2}-2^{-(k+2)}$.

Thus, if $k(n)=O(\log n)$ then the simulator fails with noticeable probability in both cases, and so the Item (2-a) follows. Item (2-c) follows merely by Case 1, since for strict simulation we have $p=1$. Item (2-b) follows by carrying out the entire argument assuming that the simulator makes only $k-2$ queries. The conclusion will be that the simulator produces the right answer with probability at most $\frac{p}{4}\leq\frac{1}{4}$, which is a clear failure. Part (2) follows. □

**Proof of Part 4:** Here we show how to foil all oracle machines which make at most $k(|x|)-2-\frac{1}{p(|x|)}$ queries *on the average*, for any polynomial $p$. Again, consider an enumeration of all probabilistic polynomial time oracle machines, and select a different input $x$ for each machine $M$. Let $k=k(|x|)$ and $p=p(|x|)$. Let $\mu_\alpha$ denote the random variable that represents the number of queries that $M$ makes on input $x$ and oracle $\alpha$. To show that $E(\mu_\alpha)$ is "big", (i.e., bigger then $k-2-\frac{1}{p}$), we will first show that if $M^\alpha(x)$ simulates the protocol then:

**(1)** For all $\log(k \cdot p) \leq i < k$, $\text{Prob}(\mu_\alpha < k - i) < \frac{1}{k \cdot p}$

and

**(2)** For all $0 \leq i \leq \log(k \cdot p)$, $\text{Prob}(\mu_\alpha < k - i) < 2^{-i}$.

Finally, we will show that these two properties of $\mu_\alpha$ imply $E(\mu_\alpha) > k - 2 - \frac{1}{p}$.

We begin by showing Property (1). Assume, by way of contradiction, that there exists an $i$ such that $\log(k \cdot p) \leq i < k$ and $\text{Prob}(\mu_\alpha < k - i) \geq \frac{1}{k \cdot p}$. Consider the behavior of $M$ after making $k - i - 1$ queries to the oracle $\alpha$. Machine $M$ may either query more, or halt outputting a string without making further queries. By hypothesis, the later happens with probability at least $\frac{1}{k \cdot p}$. We use this polynomial fraction of the output distribution to foil the simulation; namely, we show that for an appropriately chosen $K(x)$, half of this (polynomial) fraction of the output distribution is different from $K(x)$ no matter what the oracle set is. Again, consider the output distribution of $M$ on input $x$ and access to a *random* oracle $\alpha'$. By the pigeon hole principle, there is a string $y$ of length $k$, that appears in this distribution with probability less or equal to $2^{-k}$. Again, a random choice of $k - i - 1$ answers agrees with the answers of any specific oracle set $\alpha$ with probability $\frac{1}{2^{k-i-1}}$. Therefore, for any specific oracle $\alpha$, the string $y$ appears in the output distribution of $M^\alpha(x)$ with probability at most $2^{-k} \cdot 2^{k-i-1} = 2^{-i-1}$. Selecting $K(x) = y$, we get that for every oracle $\alpha$ the probability that $M^\alpha(x)$ does not output $K(x)$ is at least $\frac{1}{k \cdot p} - 2^{-i-1}$. Using $i \geq \log(k \cdot p)$ we get

$$\text{Prob}(M^\alpha(x) \neq K(x)) \;\geq\; \frac{1}{k \cdot p} - 2^{-1-\log(k \cdot p)} \;=\; \frac{1}{2 \cdot k \cdot p} \;=\; \frac{1}{\text{poly}(n)}$$

which implies that $M$ fails to simulate $\Pi$ on $x$. Thus, if $M$ simulates the transcript of the protocol properly then Property (1) must hold. A similar argument for $0 \leq i \leq \log(k \cdot p)$ shows that we can choose $K(x) = y$ so that $M$ fails to simulate on input $x$ with probability at least $2^{-i} - 2^{-i-1} = 2^{-i-1}$. Using $i \leq \log(k \cdot p)$ we again get $\text{Prob}(M^\alpha(x) \neq K(x)) \;\geq\; \frac{1}{2k \cdot p} \;=\; \frac{1}{\text{poly}(n)}$, which implies that $M$ fails to simulate $\Pi$ on $x$. Thus, if $M$ simulates the transcript of the protocol properly then Property (2) must hold too.

Clearly,

$$\begin{aligned}
E(\mu_\alpha) &\geq \sum_{j=0}^{k} \text{Prob}(\mu_\alpha = j) \cdot j \\
&= k - \sum_{i=1}^{k} \text{Prob}(\mu_\alpha = k - i) \cdot i \\
&= k - \sum_{i=1}^{k} \text{Prob}(\mu_\alpha \leq k - i) \\
&= k - \sum_{i=0}^{k-1} \text{Prob}(\mu_\alpha < k - i)
\end{aligned}$$

Using Properties (1) and (2) we get:

$$\begin{aligned}
E(\mu_\alpha) &\geq k - \sum_{i=0}^{\log(k \cdot p)-1} \text{Prob}(\mu_\alpha < k - i) - \sum_{i=\log(k \cdot p)}^{k-1} \text{Prob}(\mu_\alpha < k - i) \\
&> k - \sum_{i=0}^{\log(k \cdot p)-1} 2^{-i} - \sum_{i=\log(k \cdot p)}^{k-1} \frac{1}{k \cdot p}
\end{aligned}$$

$$> \quad k - 2 - \frac{1}{p}$$

Part (4) follows. $\Box$

## 5.2 Gaps between some measures

In this subsection, we present protocols which demonstrate the difference between the various measures. Specifically, we investigate the gap between the oracle and the average oracle measures, between the hint and the strict oracle measures, and between the oracle and the strict oracle measures. Finally, we consider the variant of the oracle definition in which the simulator is required to produce an output with probability at least $\delta$ for some $0 < \delta < 1$ which is not necessarily $1/2$. This variant was related to the oracle measure in Propositions 2.6 and 2.7. We present protocols which demonstrate that the relations shown in Propositions 2.6 and 2.7 are (almost) tight.

Let us begin by showing that the gap between the average oracle and the other oracle versions, cannot be bounded, even by a polynomial.

**Proposition 5.2** (the AVERAGE oracle measure may be much lower than the strict one): *For every polynomial time computable $k(|x|)$ and every non-negligible (and polynomial-time computable) fraction $p(|x|)$ (i.e., $\exists c > 0$ such that $p(|x|) > \frac{1}{|x|^c}$) there exists an interactive proof system (for $\{0,1\}^*$) satisfying:*

1. *$(P,V)$ has knowledge complexity at most $p(|x|) \cdot k(|x|) \; < \; 1$ in the* average *Oracle sense.*
2. *$(P,V)$ has knowledge complexity greater than $k(|x|) - 2 - \log \frac{1}{p(|x|)}$ in the Oracle sense.*

**Proof:** We consider an interactive proof in which, on input $x$, with probability $p(|x|)$ the prover sends $K(x)$ to the verifier (and otherwise sends nothing). Clearly, for $|K(x)| \leq k(|x|)$, this interactive proof has *average* knowledge complexity at most $p(|x|) \cdot k(|x|)$ (i.e., in the average Oracle sense). We now show that for an appropriate choice of $K$, the above interactive proof has knowledge complexity greater than $t(|x|) \stackrel{\text{def}}{=} k(|x|) - 2 - \log \frac{1}{p(|x|)}$ in the Oracle sense. As in the proof of Theorem 5.1, we make a mental experiment considering the output of an arbitrary oracle machine $M$ on input $x$ and access to a *random* oracle $\alpha$. We set $K(x) = y$, where $y$ is a $k(|x|)$-bit string satisfying $\text{Prob}(M^{\alpha}(x) = y) \leq 2^{-k(|x|)}$ (for a random oracle $\alpha$). As each oracle $\alpha$ has probability $2^{-t(|x|)}$ to coincide with $t(|x|)$ random answers, it follows that for any specific oracle set $\alpha$, we have

$$\text{Prob}(M^{\alpha}(x) = y) \leq 2^{t(|x|)} \cdot 2^{-k(|x|)} = 2^{-2-\log_2(1/p(|x|))} = \frac{p(|x|)}{4}$$

On the other hand, the protocol $(P,V)$ outputs $y$ with probability at least $p(|x|)$. Since $p(|x|)$ is a non-negligible fraction, it follows that $M$ (even when allowed to produce no output with probability $1/2$) fails to simulate the protocol $(P,V)$ (no matter which oracle is used to answer $M$'s queries). The proposition follows. $\Box$

Interestingly, we can upper bound the knowledge complexity of the protocol appearing in the above proof by $k(|x|)+1-\log\left(\frac{1}{p(|x|)}\right)$ even in the strict oracle measure. The simulator (for the strict oracle version) guesses the first $\log \frac{1}{p(|x|)}$ bits of $K(x)$ and sends them to the oracle, which replies whether this guess is correct or not. In case it is not, the simulator outputs nothing. Otherwise, it queries the oracle for the remaining $k(|x|) - \log \frac{1}{p(|x|)}$ bits of $K(x)$. Thus, the simulator makes at most $k(|x|) - \log\left(\frac{1}{p(|x|)}\right) + 1$ queries, and output $K(x)$ with probability $p(|x|)$ as it should.

The following proposition shows a huge gap between the hint version and the strict oracle version of knowledge complexity.

**Proposition 5.3** (the hint measure may be much higher than the oracle measure): *There exists an interactive proof* $\Pi$ *that has knowledge complexity 1 in the strict Oracle sense, yet for any polynomial* $p$*, it holds that* $\mathbf{kc}_{\rm hint}(\Pi) > p(|x|)$.

**Proof:** The protocol $\Pi$ is the following: $V$ sends a random string $r$ in $\{0,1\}^{|x|}$ to $P$, which responds with a single bit $K(x,r)$. Clearly, for any $K$, this protocol yields one bit of knowledge in the strict oracle sense. Let $n = |x|$. We build a function $K(x,r)$ such that any probabilistic polynomial time machine that uses a hint of length less than $t(n) \stackrel{\rm def}{=} \frac{1}{10} \cdot 2^n$ fails to simulate $\Pi$. We consider an enumeration of all probabilistic polynomial time "hint machines" and for each machine $M$ we select a different input $x$, on which $M$ fails to simulate $\Pi$. Let us consider the distributions $M(x,h)$ for all possible hints $h \in \{0,1\}^{t(n)}$. For each $h \in \{0,1\}^{t(n)}$, define a function $f^{(h)} : \{0,1\}^n \to \{0,1\}$ so that $f^{(h)}(r) = \sigma$ iff $\text{Prob}(M(x,h) = (r,\sigma)) \geq \text{Prob}(M(x,h) = (r,\bar{\sigma}))$ (set $\sigma = 0$ if the probabilities are equal). Clearly, $M(x,h)$ does not simulate the protocol properly if $f^{(h)}(\cdot)$ is not "close" to $K(x,\cdot)$. We show that there is a function that is not close to any $f^{(h)}$ ($h \in \{0,1\}^{t(|x|)}$), and thus, by selecting this function for $K(x,\cdot)$, we foil the simulation of $M$ on $x$ (no matter which $h$ is used as a hint). We say that a function $f$ is close to a function $g$ if they disagree on less than $\frac{1}{6}$ of their possible inputs. Using this notation, if $f^{(h)}(\cdot)$ is not close to $K(x,\cdot)$ then for at least $\frac{1}{6}$ of the $r \in \{0,1\}^n$ it holds that

$$\text{Prob}\left(\; M(x,h) \;\neq\; (r, K(x,r)) \;\mid\; M(x,h) = (r,\cdot)\,\right) \;\geq\; \frac{1}{2}$$

Since $(P,V)$ outputs $(r, K(x,r))$ for a uniformly selected $r$, we get that the statistical difference between the distributions $M(x,h)$ and $(P,V)(x)$ is at least $\frac{1}{12}$. Thus, $M$ does not simulate $(P,V)$ properly (with hint $h$).

We use a counting argument to show that there exists a function $K(x,\cdot) : \{0,1\}^n \to \{0,1\}$ which is not close to any of the possible $f^{(h)}$'s. The number of possible $f^{(h)}$'s is $2^{t(n)}$ (this is the number of possible $h$'s), and the number of functions that agree with a specific $f^{(h)}$ on at least $\frac{5}{6}$ of the $r$'s is bounded by $l(n) \stackrel{\rm def}{=} \binom{2^n}{\frac{2^n}{6}} \cdot 2^{\frac{2^n}{6}}$. Therefore, the total number of functions that are close to $f^{(h)}$ for some $h \in \{0,1\}^{t(n)}$ is less than $l(n) \cdot 2^{t(n)} < 2^{2^n}$, and we can select a function $K(x,\cdot)$ to foil the simulation of $M$ on $x$. $\Box$

It was asserted in Proposition 4.7 that for any interactive proof $\Pi$, and any unbounded $g : \mathsf{N} \to \mathsf{N}$, $\mathbf{kc}^{\rm strict}_{\rm oracle}(\Pi) \leq \mathbf{kc}^{1/2}_{\rm oracle}(\Pi) + \log(\log(|x|)) + g(|x|)$. The following proposition shows that we can not do better in general.

**Proposition 5.4** (tightness of Proposition 4.7): *For every polynomial time computable function* $k : N \to N$ *and for any constant* $c > 0$*, there exists an interactive proof* $\Pi$ *such that:* $\mathbf{kc}^{1/2}_{\rm oracle}(\Pi) = k(|x|) + 1$ *and* $\mathbf{kc}^{\rm strict}_{\rm oracle}(\Pi) \geq k(|x|) + \log(\log(|x|)) + c$.

**Proof:** Suppose, first, that $k(|x|) = 0$. Let $S_x \subseteq \{0,1\}^{|x|}$ be a set of cardinality $\frac{1}{2} \cdot 2^{|x|}$. Consider a protocol, $\Pi$, in which the prover $P$ sends $V$ a uniformly chosen $y \in S_x$. A simulator of the oracle type can choose a string uniformly in $\{0,1\}^{|x|}$, and ask the oracle whether $y \in S_x$. Thus, for any $S_x$, the simulator outputs a conversation of the right distribution with probability $\frac{1}{2}$. We use again the diagonalization technique, and show that for each strict oracle simulator $M$, which makes at most $\log(\log(|x|)) + c$ queries to the oracle, there exists a set $S_x$ such that $M$ can not simulate $\Pi(x)$ properly. In particular, we show that even if $S_x$ is uniformly chosen among the subsets of $\{0,1\}^{|x|}$ which have cardinality $\frac{1}{2} \cdot 2^{|x|}$, the probability that $M$ does not output an element of $S_x$ is non-negligible (no matter which oracle is used to answer $M$'s queries). Thus, there exist such an $S_x$ which $M$ fails to simulate. As before, we denote by $M_r^{\alpha}(x)$ the output of $M$ on input $x$, coin tosses $r$ and oracle $\alpha$. First, we show that for a randomly chosen set $S_x$ and any *fixed* random string $r$, the probability that $M$ does not output an element of $S_x$ is non-negligible no matter what the oracle answers are. It follows that the probability that $M$ does not output an element of $S_x$, when $S_x$ is random as before, is non-negligible also when the string $r$ is uniformly chosen.

First, note that (for a fixed $r$) the set $\{M_r^{\alpha}(x)\}_{\alpha}$ contains only $O(\log(|x|))$ different strings. This follows from the fact that the behavior of $M_r$ is completely determined by the $\log(\log(|x|)) + c$ bits which it gets from the oracle, and there are only $O(\log(|x|)$ possible different sequences of answers. Therefore, the probability that all the strings in $\{M_r^{\alpha}(x)\}_{\alpha}$ are not in $S_x$ (for a randomly chosen $S_x$) is at least $\left(\frac{1}{2}\right)^{O(\log(|x|))} = \frac{1}{|x|^{O(1)}}$. In other words, the probability that machine $M$ does not output an element of $S_x$ is greater than a polynomial fraction. Since the argument holds for any $r$, it certainly holds for a random $r$, leading to the conclusion, that for any oracle $\alpha$ (and a random $S_x$) the probability that $M$ does not output an element in $S_x$ is greater than a polynomial fraction. Thus, there exists such $S_x$, and we can use it to foil $M$ on $x$.

To extend the proof for any polynomial computable function $k(|x|)$, we compose (sequentially) the above protocol with the protocol used in the proof of Theorem 5.1. It seems intuitively clear that this composition yields the desired assertion and this intuition is indeed valid. A complete proof has to combine both proofs into a single diagonalization argument. The straightforward details are omitted. □

Propositions 2.6 and 2.7 state the relations between the oracle measure, and the measures we get by changing the constant 1/2 (the lower bound on the probability that the simulator produces an output) to any constant $0 < \delta < 1$. The following propositions show that we cannot get tighter relations in general.

**Proposition 5.5** (tightness of Proposition 2.7): *For any polynomial computable function $k : N \to N$, and for each constant $0 < \epsilon < \frac{1}{2}$, there exists a protocol $\Pi_\epsilon$ such that $\mathbf{kc}_{\text{oracle}}^{1/2}(\Pi_\epsilon) = k(|x|) + 1$ and $\mathbf{kc}_{\text{oracle}}^{1-\epsilon}(\Pi_\epsilon) \geq k(|x|) + \log(\log(\frac{1}{\epsilon}))$.*

**Proof:** By the same construction as in the previous proof. □

**Proposition 5.6** (tightness of Proposition 2.6): *For every interactive proof $\Pi$, and every $0 < \epsilon < \frac{1}{2}$, $\mathbf{kc}_{\text{oracle}}^{\epsilon}(\Pi) \leq 1 + \max\{\mathbf{kc}_{\text{oracle}}^{1/2}(\Pi) - \lfloor\log(\frac{1}{2\epsilon})\rfloor,\ 0\}$. In particular, for every polynomial time computable function $k : N \to N$ which is bounded by some fixed polynomial, there exists an interactive proof $\Pi$ such that $\mathbf{kc}_{\text{oracle}}^{\epsilon}(\Pi) = k(|x|)$ and $\mathbf{kc}_{\text{oracle}}^{1/2}(\Pi) \geq k(|x|) + \log(\frac{1}{2\epsilon}) - 2$.*

The second part of the above proposition is actually the one which asserts the tightness of Proposition 2.6.

**Proof:** We show how to transform a simulator which produces output with probability at least 0.5 into one which uses less queries but produces output with probability at least $\epsilon < 0.5$. The existence of the protocol $\Pi$ mentioned in the second assertion follows by applying the above transformation to a protocol of knowledge complexity at least $k(|x|)+\lfloor\log(\frac{1}{2\epsilon})\rfloor-2$ and at most $k(|x|)+\lfloor\log(\frac{1}{2\epsilon})\rfloor-1$ in the oracle sense. (Such a protocol exists by the strictness of the oracle hierarchy – see Theorem 5.1.)

Given a simulator which produces output with probability at least 0.5, we construct a new simulator as follows. The new simulator produces a random string of coins, denoted $r$, for the original simulator, and guess at random the first $\lfloor\log(\frac{1}{2\epsilon})\rfloor$ answers, denoted $\alpha$, that the original oracle would have provided (the original simulator running under these coins). Next, it queries the (augmented) oracle on whether this prefix of the oracle answers (i.e., $\alpha$) is correct for the chosen random string (i.e., $r$) and whether the original simulator produces an output on this random string. That is, the query is a pair $(r,\alpha)$ and the answer is one bit. If the answer is YES, then the new simulator invokes the original simulator on coins $r$, answering the first $|\alpha|$ queries by the bits of $\alpha$ and referring the subsequent queries to the oracle. Otherwise, the new simulators halts with no output.

With probability at least $\frac{1}{2}$, the original simulator produces an output when given access to a good oracle. The new simulator correctly guesses the first $\lfloor\log(\frac{1}{2\epsilon})\rfloor$ responses of this oracle with probability at least $2\epsilon$. Thus, the probability that the new simulator produces an output is at least $\frac{1}{2}\cdot 2\epsilon = \epsilon$. □

### 5.3 Knowledge complexity is not preserved under sequential repetitions

Recall that knowledge complexity in the hint sense is preserved under sequential repetitions (cf., Proposition 3.3). In contrast, we note that knowledge complexity in any of the other measures discussed above is not preserved under sequential repetitions. This holds even if we restrict our attention only to honest verifiers (see Subsection 3.5). For example –

**Proposition 5.7** *There exists an interactive proof* $\Pi$ *which has knowledge complexity 1 in the* (strict) *oracle sense while* $\Pi^2$ *has knowledge complexity at least* $2-1/\text{poly}(n)$ *in the fraction sense as well as in the average oracle sense* (even when restricting attention to the honest verifier).

It follows that $\Pi^2$ has has knowledge complexity 2 in the oracle sense. Recall that $\Pi^2$ denotes the interactive proof system in which the system $\Pi$ is repeated twice.

**Proof outline:** We use a proof system $\Pi$ analogous to the one in the proof of Proposition 5.3: Specifically, the verifier randomly selects a bit $r\in\{0,1\}$, sends it to the prover which responds with a single bit $K(x,r)$, where $K:\{0,1\}^*\times\{0,1\}\mapsto\{0,1\}$ is defined below. Clearly, $\mathbf{kc}^{\text{strict}}_{\text{oracle}}(\Pi)\le 1$ (as well as $\mathbf{kc}^{\text{strict}}_{\text{oracle}}(\Pi^2)\le 2$). To establish the lower bound, we use again the diagonalization technique. Specifically, using the average oracle measure, we need to fail all simulators which with some noticeable (i.e., $1/\text{poly}(n)$) probability makes only a single query. Note that with probability 1/4 the honest verifier sends the bit 0 in the first run and the bit 1 in the second. Thus, the situation reduces to the one analyzed in the proof of Theorem 5.1, and so we may apply the same arguments here (i.e., to the function $K(x)\stackrel{\text{def}}{=}(K(x,0),K(x,1))$). □

## 6 Knowledge complexity of languages in the hint sense

In this section we investigate the "hint-knowledge complexity" hierarchy of languages and establish two results - $\mathcal{KC}_{\text{hint}}(poly(|x|))\subseteq\mathcal{AM}[2]$, and $\mathcal{KC}_{\text{hint}}(O(\log(|x|))\subseteq\text{co}\mathcal{AM}[2]$. These results are obtained by extending the result proven for zero-knowledge by Fortnow [10] and Aiello and Håstad

[2]. In the sequel, we follow the construction of [2][20]. One doesn't have to master the techniques used in that work in order to understand our proofs. Yet, some properties of these techniques, explicitly stated below, are essential to the validity of our proofs.

The construction in [2] considers the interactive proof $(P, V)$ for $L$ and the simulator $M$ of $(P, V)$ guaranteed by the hypothesis[21]. They use the simulator to build a new interactive proof $(P', V')$ for $L$ which is of constant number of rounds. A simple enhancement in the construction (see [2, 22]) produces also an interactive proof $(P'', V'')$ for $\overline{L}$ (the complement of $L$) which also has a constant number of rounds. (Employing [21] and [5] they get that $L$ and $\overline{L}$ are in $\mathcal{AM}[2]$.)

We first note that the use of $M$ in these proof systems is limited. The proof considers only the function $f_{M,x}$ which is defined so that $f_{M,x}(r)$ is the output of $M$ on input $x$ and random string $r$. The simulator is not considered on other inputs, and the algorithm by which $f_{M,x}(\cdot)$ is computed is immaterial as long as it runs in polynomial time in $|x|$.

By the definition of the simulator $M$, there is no restriction on its behavior when the input $x$ is not in $L$ (except for being polynomial time). Namely, when $x \notin L$, the only property of $f_{M,x}(\cdot)$ which is guaranteed by $M$ being a simulator is that $f_{M,x}(\cdot)$ is computable in polynomial time (in $|x|$).

Returning to the proof systems in [10, 2], we get that when $x \notin L$, the properties of these proof systems are maintained also in the case that they are given access to *any* polynomial time (in the length of their input $x$) computable function $f(\cdot)$ and not necessarily to $f_{M,x}(\cdot)$. Thus, we get

**Claim 6.1** *Suppose the protocol $(P', V')$ (resp. $(P'', V'')$) is given access to some arbitrary probabilistic polynomial time machine $M'$ instead of the simulator $M$. Then, it still holds that for any $x \notin L$ the protocol $(P', V')$ accepts (resp. $(P'', V'')$ rejects) $x$ with negligible probability (as it would have accepted (resp. rejected) with access to the original simulator).*

**Comment:** Both protocols require that the original interactive proof $(P, V)$, has an exponentially small error probability. This causes no difficulty when using the *honest* verifier in the Hint version. We can run several copies of $(P, V)$ in parallel without increasing the knowledge complexity, since all copies of the simulation can use the same hint.

**Theorem 6.2** *Let $L$ be a language that has an interactive proof with knowledge complexity $k = poly(|x|)$ in the Hint sense, then $L \in \mathcal{AM}[2]$.*

**Proof:** We have a language $L$ accepted by an interactive proof $(P, V)$, and a simulator $M$ that on the input $x$, and the hint $h(x)$, produces a conversation. If $M$ gets the right hint, it produces a good simulation of $(P, V)$. Otherwise, nothing is guaranteed about the behavior of $M$, except for polynomial running time.

We use the interactive proof $(P', V')$ for $L$ given by [2] with a preliminary step. In this step, $P'$ sends $V'$ the hint $h(x)$ associated with the input $x$. After this step, $P'$ and $V'$ build a machine $M'(x) \stackrel{\text{def}}{=} M(x, h(x))$ and proceed by running the protocol $(P', V')$ on the input $x$ using the simulator $M'$.

[20] The $\mathcal{AM}[2]$ protocol built in [10] for a language $L$ whose complement has a statistical zero-knowledge interactive proof has a flaw (see Appendix A in [15] for further details). However, the basic ideas in [10] were extended in [2] to construct an $\mathcal{AM}[2]$ protocol for a language $L$ that has a statistical zero-knowledge interactive proof. Furthermore, the additional machinery presented in [2] suffices also for proving Fortnow's result (see [22] details of how to use the machinery of [2] to prove Fortnow's result). Alternatively, see [26].

[21] Note that though the zero-knowledge property implies the existence of many simulators (one for each possible verifier), [2] use only the simulator for the original interactive proof $(P, V)$, where $V$ is not cheating.

It is clear that if both prover and verifier act according to the protocol, then completeness is ensured. Claim 6.1 implies the soundness of the protocol. The number of rounds is a constant, and using [5] and [21] we get $L \in \mathcal{AM}[2]$ as desired. $\square$

**Theorem 6.3** *Let $L$ be a language that has an interactive proof with knowledge complexity $k = O(\log(n))$ in the Hint sense, then $L \in \text{co}\mathcal{AM}[2]$.*

**Proof:** Let us define $2^{k(|x|)}$ new simulators. For each $\alpha \in \{0,1\}^k$, let $M'_\alpha(x) \stackrel{\text{def}}{=} M(x, \alpha)$ where $M$ is the hint machine which simulates the original interactive proof $(P, V)$. Obviously, $M'_{h(x)}$ is a good simulating machine for $(P, V)$. The interactive proof we build runs $(P'', V'')$, the protocol constructed in [2] for $\bar{L}$, for $2^k$ times in parallel. The $i^{\text{th}}$ copy uses $M'_i$ as its black box simulator. Our new verifier will accept the input $x$ iff all the sub-protocols end up accepting.

*Completeness:* Suppose $x \in \bar{L}$ (i.e., $x \notin L$). The construction of [2] guarantees that $(P'', V'')$ accepts $x$ with probability at least $1 - \mu(|x|)$ when $P''$ and $V''$ are given access to a proper simulator and where $\mu : N \to [0,1]$ is some negligible fraction. However, by Claim 6.1, $(P'', V'')$ accepts $x$ with this probability also when $P''$ and $V''$ are given access to any probabilistic polynomial time machine (and not necessarily to a simulator for $(P, V)$). Therefore, each copy of the protocol $(P'', V'')$ rejects $x$ with a negligible probability, and the probability that at least one of these copies rejects $x$ is bounded by $2^{k(|x|)} \cdot \mu(|x|)$. Since $k(|x|) = O(\log |x|)$ and since $\mu(|x|)$ is negligible, this probability is also negligible.

*Soundness:* If $x \notin \bar{L}$ (i.e. $x \in L$) then the copy of $(P'', V'')$ which uses $M'_{h(x)}$ rejects $x$ with probability almost 1, and since our verifier accepts only if *all* the copies end up accepting, it will reject $x$ with probability almost 1. $\square$


## Acknowledgment:

We are grateful to Benny Chor, Johan Håstad, Hugo Krawczyk, and Eyal Kushilevitz for helpful discussions.


## References


[1] Aiello, W., M. Bellare, and R. Venkatesan, "Knowledge on the Average – Perfect, Statistical, and Logarithmic", *Proc. 27th STOC*, pp. 469–478, 1995.

[2] Aiello, W., and J. Håstad, "Perfect Zero-Knowledge Languages can be Recognized in Two Rounds", *JCSS*, Vol. 42, pages 327–345, 1991.

[3] W. Alexi, B. Chor, O. Goldreich, and C. P. Schnorr, "RSA/Rabin Functions: Certain Parts are As Hard As the Whole", *SIAM J. Comp.*, Vol. 17, No. 2, April 1988, pp. 194–209.

[4] Alon, N., L., Babai and A. Itai, "A Fast and Simple Randomized Algorithm for the Maximal Independent Set Problem", *J. of Algorithms*, Vol. 7, 1986, pp. 567–583.

[5] Babai, L., "Trading group theory for randomness", *Proc. 17th STOC*, 1985, pp. 421-429.

[6] Bar-Yehuda, R., B. Chor, and E. Kushilevitz, "Privacy, Additional Information, and Communication", *5th IEEE Structure in Complexity Theory*, July 1990, pp. 55-65.

[7] Bellare M. and E. Petrank, "Making Zero Knowledge Provers Efficient", *Proc. 24th STOC,* 1992, pp. 711-722.

[8] Ben-Or, M., O. Goldreich, S. Goldwasser, J. Håstad, J. Kilian, S. Micali, and P. Rogaway, "Everything Provable is Provable in Zero-Knowledge", *Advances in Cryptology - Crypto88 (proceedings),* Springer-Verlag, Lecture Notes in Computer Science, Vol. 403, pp. 37-56, 1990.

[9] T.M. Cover and G.A. Thomas, *Elements of Information Theory*, John Wiley & Sons, Inc., New-York, 1991.

[10] Fortnow, L., "The Complexity of Perfect Zero-Knowledge", *Advances in Computing Research: a research annual*, Vol. 5 (Randomness and Computation, S. Micali, ed.), pages 327–343, 1989.

[11] O. Goldreich, *Foundations of Cryptography – Fragments of a Book* Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Israel, February 1995. Available from `http://theory.lcs.mit.edu/~oded/frag.html` and `http://www.eccc.uni-trier.de/eccc/`.

[12] Goldreich, O. and H. Krawczyk, "On the Composition of Zero-Knowledge Proof Systems", *SIAM Journal on Computing*, Vol. 25, No. 1, February 1996, pp. 169–192.

[13] Goldreich, O., S. Micali, and A. Wigderson, "Proofs that Yield Nothing But their Validity or All Languages in NP Have Zero-Knowledge proof Systems", *Jour. of ACM.*, Vol. 38, 1991, pp. 691–729.

[14] Goldreich, O. and Y. Oren, "Definitions and Properties of Zero-Knowledge Proof Systems", *Jour. of Cryptology*, Vol. 7, 1994, pp. 1–32.

[15] Goldreich, O., R. Ostrovsky, and E. Petrank, "Computational Complexity and Knowledge Complexity", *26th ACM Symp. on Theory of Computation*, May 1994. pp. 534–543. To appear in *SIAM J. on Comput.*, 1998.

[16] Goldreich, O. and E. Petrank, "Quantifying Knowledge Complexity", *the 32nd Annual IEEE Symposium on the Foundations of Computer Science*, October 1991, pp. 59–68.

[17] S. Goldwasser and S. Micali, "Probabilistic Encryption", *JCSS*, Vol. 28, No. 2, pages 270–299, 1984. Preliminary version in *14th STOC*, 1982.

[18] Goldwasser, S., S. Micali, and C. Rackoff, "The Knowledge Complexity of Interactive Proofs", *Proc. 17th STOC,* 1985, pp. 291-304.

[19] Goldwasser, S., S. Micali, and C. Rackoff, "The Knowledge Complexity of Interactive Proof Systems", *SIAM Jour. on Computing,* Vol. 18, 1989, pp. 186-208.

[20] Goldwasser, S., S. Micali, and P. Tong, "Why and How to Establish a Private Code on a Public Network", In *23rd FOCS*, pages 134–144, 1982.

[21] Goldwasser, S., and M. Sipser, "Private Coins vs. Public Coins in Interactive Proof Systems", *Advances in Computing Research (ed. S. Micali),* 1989, Vol. 5, pp. 73-90.

[22] Håstad, J., Perfect Zero-Knowledge in $\mathcal{AM} \cap \text{co}\mathcal{AM}$. Unpublished (2-page) manuscript explaining the underlying ideas behind [2]. 1994.

[23] Impagliazzo, R., and M. Yung, "Direct Minimum-Knowledge Computations", *Advances in Cryptology - Crypto87 (proceedings),* Springer-Verlag, Lectures Notes in Computer Science, Vol. 293, 1987, pp. 40-51.

[24] E. Kushilevitz and N. Nisan, *Communication Complexity*, Cambridge University Press, 1996.

[25] G. L. Miller, "Riemann's Hypothesis and Tests for Primality", *JCSS*, Vol. 13, 1976, pp. 300–317.

[26] E. Petrank and G. Tardos, "On the Knowledge Complexity of NP", In *37th FOCS*, pages 494–503, 1996.

[27] Shannon, C.E., "A mathematical theory of communication", *Bell Sys. Tech. J.*, Vol. 27, 1948, pp. 623–656.

[28] A.C. Yao, "Some complexity questions related to distributive computing", In *11th STOC*, pages 209–213, 1979.

[29] A.C. Yao, "Theory and Application of Trapdoor Functions", In *23rd FOCS*, pages 80–91, 1982.