Appears in Proceedings of CRYPTO 92.

On Defining Proofs of Knowledge

Mihir Bellare* Oded Goldreich†

August 26, 1992

Abstract

The notion of a "proof of knowledge," suggested by Goldwasser, Micali and Rackoff, has been used in many works as a tool for the construction of cryptographic protocols and other schemes. Yet the commonly cited formalizations of this notion are unsatisfactory and in particular inadequate for some of the applications in which they are used. Consequently, new researchers keep getting misled by existing literature. The purpose of this paper is to indicate the source of these problems and suggest a definition which resolves them.

* High Performance Computing and Communications, IBM T.J. Watson Research Center, PO Box 704, Yorktown Heights, NY 10598, USA. e-mail: `mihir@watson.ibm.com`

† Computer Science Department, Technion, Haifa, Israel. e-mail: `oded@cs.technion.ac.il`. Research was partially supported by grant No. 89-00312 from the US-Israel Binational Science Foundation (BSF), Jerusalem, Israel.

1 Introduction

The introduction of the concept of a "proof of knowledge" is one of the many conceptual contributions of the work of Goldwasser, Micali and Rackoff [14]. This fundamental work, though containing intuition and clues towards a definition of the notion of a "proof of knowledge," does not provide a formal definition of it. Furthermore, in our opinion, the commonly cited formal definitions, namely those of Feige, Fiat and Shamir [6] and Tompa and Woll [18], are not satisfactory, and, in particular, inadequate for some of the applications in which they have been used.

The purpose of this paper is two-fold. First, we would like to describe whence stem the flaws in the previous definitions and why these definitions do not suffice for some applications. We then propose a definition which we feel remedies these defects and also has other advantages.

We note that a definition which is much better than those of [6, 18] has appeared in the work of Feige and Shamir [7], but the community seems unaware of the fact that the definition in [7] is fundamentally different from, and preferable to, the one in [6] (in particular, this fact is not stated in [7]). The definition we present differs in many ways from that of [7] which we feel still has some conceptual problems. Yet both have in common the attempt to capture provers who convince with probabilities that are not non-negligible, thereby correctly addressing what we believe is one of the main flaws in the definitions of [6, 18].

Among the novel features of our new definition is that it allows us also to talk of the knowledge of machines which operate in super-polynomial-time. But this (and other novel features) we will discuss later; let us begin with the basics.

1.1 Basic approach in defining proofs of knowledge

Intuitively, a two-party protocol constitutes a "system for proofs of knowledge" if "whenever" one party (called the verifier) is "convinced"[1] then the other party (called the prover) indeed "knows" "something". The excessive use of quotation symbols in the condition of the above statement may provide some indication to the complexity of the notion. For simplicity, let us consider the special case in which the "object of knowledge" is a witness for membership of a common input in some predetermined language in NP. For example, let us consider the case in which the "object of knowledge" is a satisfying assignment for a CNF formula (given as input to both parties). Hence, a two-party protocol constitutes a "system for proofs of knowledge of satisfying assignments" if "whenever" the verifier is "convinced" then the prover indeed "knows" a satisfying assignment for the given formula. The clue to a formalization of "proofs of knowledge" is an appropriate interpretation of the phrases "whenever" and "knows" which appear in the condition. The phrase "convinced" has the straightforward and standard interpretation of accepting (i.e., entering a specified state in the computation).

Following [14] the interpretation of the phrases "whenever" and "knows" is as follows. Suppose for simplicity that the verifier is always convinced (i.e. after interaction with the prover the verifier always enters an accepting state). Saying that the prover "knows" a satisfying assignment means that it "can be modified" so that it outputs a satisfying assignment. The notion of "possible modifications of machine M" is captured by efficient algorithms that use M as an oracle. Hence, saying that the prover "knows" a satisfying assignment means that it is feasible to compute a satisfying assignment by using the prover as an oracle. Namely, there exists an efficient algorithm, called the *knowledge extractor*, that on input a formula ϕ and given oracle access to a good prover (i.e. a prover which always convince the verifier on common input ϕ) is able to output a satisfying

[1] We have replaced the more intuitive but possibly misleading phrase "convinced that the prover knows something" by the neutral phrase "convinced".

assignment to ϕ. Indeed, this is exactly the interpretation given in works as [18, 6]. The problem is to deal with the general case in which the prover may convince the verifier with some probability $\epsilon < 1$. Again, for constant ϵ there is no problem and it can be required that even in this case the knowledge extractor succeeds in outputting a satisfying assignment in expected polynomial-time (or alternatively output such an assignment in polynomial time with probability exponentially close to 1). This interpretation is valid also if ϵ is any non-negligible function of the length of the input ϕ (a non-negligible function in n is a function which is asymptotically bounded from below by a function of the form n^{-c}, for some constant c). But *what should be required if the prover does not convince the verifier with non-negligible probability*? Most previous formulations (e.g., [18, 6]) require nothing, and hence are unsatisfactory both from a conceptual point of view and from a practical point of view (i.e., in view of many known applications). In particular, this inadequacy often appears when "proofs of knowledge" are used as subprotocols inside larger protocols. In other words, the inadequate formulations of "proofs of knowledge" drastically limit their modular application in the construction of cryptographic protocols.

1.2 Provers which convince with probability that is not non-negligible

We start with an abstract justification of our claim that requiring nothing, in case the prover does not convince the verifier with non-negligible probability, is wrong. We first uncover the reason it has been believed that it is justified to require nothing. It has been believed that events which occur with probability which is not non-negligible can be ignored, just as events which occur with negligible probability can be ignored. However, a key observation, which has been overlooked by this argument, is that a sequence of probabilities can be neither negligible (i.e., smaller that n^{-c} for all $c > 0$ and all sufficiently large n's) nor non-negligible (i.e., bigger that n^{-c} for some $c > 0$ and all sufficiently large n). Hence, even if it were justified to require nothing in case the prover convinces the verifier with negligible probability, it is unjustified to require nothing in case the probability of being convinced is just not non-negligible!

To demonstrate what is wrong when we require nothing in case the prover does not convince the verifier with a non-negligible probability, we consider the following possibility. Suppose that there exist a prover and an infinite sequence of CNF formulae, $\{\phi_n : n \in \mathsf{N}\}$, such that the probability that the prover convinces the verifier on common input ϕ_n is n^{-k}, where n is the length of ϕ_n and k is the number of literals in the longest clause of ϕ_n. Furthermore, suppose that, for every $k > 0$, there exists infinitely many n's such that k is the number of literals in the longest clause of ϕ_n. An important observation is that the sequence of probabilities (defined by the above prover and formulae) is neither negligible (i.e., smaller that n^{-c} for all $c > 0$ and all sufficiently large n's) nor non-negligible (i.e., bigger that n^{-c} for some $c > 0$ and all sufficiently large n). Hence, previous definitions of "proof of knowledge" require nothing (or too little) with respect to the above prover. To appreciate the severity of the lack of requirement with respect to the above prover consider the following application. Suppose that each ϕ_n has a unique satisfying assignment, and that a "proof of knowledge of a satisfying assignment" is used as a subprotocol inside a protocol in which `Alice` will send `Bob` a satisfying assignment to ϕ_n if she is convinced by `Bob` that he already knows this assignment. We would like to argue that in this application `Alice` yields no knowledge to `Bob` (i.e., `Alice` is zero-knowledge). Using a reasonable definition of "proof of knowledge" one should be able to prove such a statement (and indeed using our definition such a proof can be presented). Yet, the zero-knowledge property of `Alice` can not be demonstrated using previous formulations of "proof of knowledge."[2]

[2]Typically, the simulator for the zero-knowledge property uses the knowledge extractor (for the proof of knowledge) as a subroutine. However, previous formulations of "proof of knowledge" do not guarantee a knowledge extractor

A more concrete and practical setting can help to further clarify our point. It has been suggested to use a "proof of knowledge" as a subprotocol inside a multi-round encryption scheme secure against chosen ciphertext attack (cf. [8, Sec. 5] and [15, Sec. 5.4]). Namely, the decryption module returns a decryption of a chosen ciphertext only if "convinced" that the party asking for it already "knows it". (This is a special case of the application considered in the previous paragraph). Using previous formalizations of "proof of knowledge" it cannot be proved that the above "decryption module" is zero-knowledge (i.e., yields no knowledge) under a chosen ciphertext attack. Yet, the above decryption module is zero-knowledge and this zero-knowledge property (though not proven!) has been used to claim that the particular multi-round encryption scheme is secure against chosen message attack. We stress that the above mentioned encryption scheme is indeed secure under such attacks, it is just that its security has not been proven but rather "hand-waved", and that the essential flaw in the hand-waving is the fact that it is based on an inadequate formalization of proofs of knowledge.

The above example is very typical. In many (yet not all) applications of "proofs of knowledge" one relies on their meaningfulness with respect to arbitrary behavior of the prover. Yet as pointed out above, previous formalizations of "proof of knowledge" are meaningful only in case the prover convince the verifier with non-negligible probability. One should not make the mistake of saying that events which happen with probability that is not non-negligible can be ignored, since such probabilities are not negligible! Put in other words, negligible is not the negation of non-negligible!

To avoid confusion we stress that the definitions of [6] do suffice for the applications in their paper. Problems (as illustrated above) have arisen when these same definitions have (later) been used in other applications.

1.3 A few words about the definition presented in this paper

The most important aspect in which our definition (as well as the one of [7]) deviates from the previous ones is that there is no sharp distinction between provers based on whether they convince the verifier with non-negligible probability or not. In our case, the requirement is that the knowledge extractor always succeeds and that the average number of steps it performs is inversely proportional (via a polynomial factor) to the probability that the prover convinces the verifier.

Over and above this change, we have taken the opportunity to correct what we feel are other conceptual drawbacks of previous definitions (including [7]). Although these other changes are to some extent a matter of taste they are nonetheless important, and also enable us to obtain definitions that are more general than previous ones. As examples, a few such issues are discussed below; we refer the reader to §4 for more details as well as for a discussion of the many other points of difference.

All previous definitions refer only to provers which can be implemented by probabilistic, polynomial time programs (with auxiliary input). In some works it is even claimed that it makes no sense to talk of the knowledge of computationally unrestricted machines. We strongly disagree with such claims, and point out that previous definitions have considered only computationally restricted provers because of technical reasons. From a conceptual point of view it is desirable to have a "uniform" definition of proofs of knowledge which refers to all provers independently of their complexity, the probability they lead the verifier to accept, and so on. In fact, our definition has this property. A consequence of this property is that our definition enables one to talk of the "knowledge" of super-polynomial-time machines. For example, we are able to say in what sense the

which handles the entire sequence of formulae. On the other hand, one cannot ignore the case in which something is sent by `Alice` since this case is not negligible.

interactive proofs introduced by Shamir [17], in order to demonstrate that IP=PSPACE, constitute "proofs of knowledge."

Most proofs of knowledge (e.g., the proof of knowledge of an isomorphism used by [12] – see Appendix E) are constructed by iterating some "atomic" protocol. Typically, these atomic protocols have the property that one can easily lead the verifier to accept with some constant probability (say, 1/2) even when having no "knowledge" whatsoever. Yet, these atomic protocols do prove some "knowledge" of the prover, in case it is able to convince the verifier with higher probability. However, previous definitions of "proof of knowledge" were unable to capture this phenomenon; they were only able to say what it means for sufficiently (i.e. super-logarithmic) many iterations of these "atomic" protocols to be "proofs of knowledge." This belies the basic intuition and also precludes a modular approach to protocol design. We correct these weaknesses by showing how to measure the "knowledge error" of a proof, and then showing how composition reduces it.

A special case of our definition is when the knowledge error is zero. This special case is important is some applications. In particular, "proofs of knowledge with zero error" are important when using a proof of knowledge inside a zero-knowledge protocol so that one party sends some information only if he is convinced that the other party already knows it. A typical example is the zero-knowledge protocol for graph non-isomorphism of [12] (cf. §7.1). We stress that none of the previous definitions could handle "proofs of knowledge with zero error."

1.4 Organization

The main conventions used throughout the paper appear in §2. The new definition (of a proof of knowledge) appears in §3, and §4 contains a discussion of various aspects of this definition. This main part of the paper is augmented by Appendix A, in which previous definitions (of proofs of knowledge) are reviewed, and by §7 in which examples of the applications of the new definition are presented.

The rest of the paper addresses issues which are related to the definition of a proof of knowledge: §5 addresses the effect of repeating a proof of knowledge, and §6 presents an equivalent formulation of our definition of a proof of knowledge.

2 Preliminaries

Let $R \subseteq \{0,1\}^* \times \{0,1\}^*$ be a binary relation. We say that R is *polynomially bounded* if there exists a polynomial p such that $|y| \leq p(|x|)$ for all $(x,y) \in R$. We say that R is an NP relation if it is polynomially bounded and, in addition, there exists a polynomial-time algorithm for deciding membership in R.

If R is a binary relation we let $R(x) = \{y : (x,y) \in R\}$ and $L_R = \{x : \exists y \text{ such that } (x,y) \in R\}$. If $(x,y) \in R$ then we call y a *witness* for x.

The proof systems we define are two-party protocols. We model the players in these protocols not (as is common) as interactive machines, but rather as what we will call "interactive functions." The idea is to separate the computational aspect of the player from its input/output behaviour. We feel that this eases and clarifies the presentation of the (later) definitions.

Definition 2.1 *An* interactive function *A associates to each $x \in \{0,1\}^*$ (common input) and $\eta \in \{0,1\}^*$ (prefix of a conversation) a probability distribution on $\{0,1\}^*$ which we denote by $A_x[\eta]$. We denote by $A_x(\eta)$ an element chosen at random from this distribution.*

Intuitively, $A_x(\eta)$ is A's next message when the prefix of the conversation so far was η and the common input is x.

The two players in the protocols we will consider are called the *prover* and the *verifier*. Both are modeled as interactive functions. The interaction between prover P and verifier V on a common input x consists of a sequence of "moves" in each of which one player sends a message to the other. The players alternate moves, and for simplicity we will assume the prover moves first and the verifier last. We denote by α_i (resp. β_i) the random variable which is the message sent by the prover (resp. verifier) in his i-th move. We assume any prefix of a conversation can be uniquely parsed into its constituent messages. Then each message is specified by the prescribed interactive function as a function of the common input and previous messages. More precisely,

$$\begin{aligned} \alpha_i &= P_x(\alpha_1\beta_1 \ldots \alpha_{i-1}\beta_{i-1}) & (i = 1, 2, \ldots) \\ \beta_i &= V_x(\alpha_1\beta_1 \ldots \alpha_{i-1}\beta_{i-1}a_i) & (i = 1, 2, \ldots)\,. \end{aligned}$$

These random variables are defined over the probabilistic choices of both interactive functions.

We will adopt the convention that there are special symbols which an interactive function may output to indicate things like acceptance or rejection. We assume there exists a function $t_V(\cdot)$ (the number of "rounds") such that the $t_V(x)$-th move of the verifier contains its verdict on acceptance or rejection. (For simplicity we restrict the number of rounds to be a function of the verifier and the common input, and do not allow it to depend on the prover. Yet this is without loss of generality). The *transcript of the interaction*, denoted $\mathrm{tr}_{P,V}(x)$, is the string valued random variable which records the conversation up to the verifier's verdict. That is, $\mathrm{tr}_{P,V}(x) = \alpha_1\beta_1 \ldots \alpha_{t_V(x)}\beta_{t_V(x)}$. Note that the transcript of the interaction between a prover P and verifier V contains the sequence of message exchanged during the interaction, but not information which is available only to one party, such as its "auxiliary input" or its "internal coin tosses," unless these were sent to the other party.

Since we have assumed that the transcript contains the verifier's verdict on whether to accept or reject, we may, for each x, talk of the set of *accepting transcripts*, denoted $\mathtt{ACC}_V(x)$, and the set of *rejecting transcripts*, denoted $\mathtt{REJ}_V(x)$. Thus the "probability that the verifier accepts" is, by definition, $\Pr[\mathrm{tr}_{P,V}(x) \in \mathtt{ACC}_V(x)]$.

We stress that the definition of an interactive function makes no reference to its computational aspects. We may discuss the computational complexity of an interactive function in a natural way, namely by the complexity of a (probabilistic) Turing machine that computes it. In particular, we say that an interactive function A is computable in probabilistic polynomial time if there exists a probabilistic Turing machine which on input x, η outputs an element distributed uniformly in $A_x[\eta]$, and runs in time polynomial in the length of x.

For simplicity we will restrict the verifier's program to be computable in probabilistic, polynomial time. (We stress that we do not restrict the computational power of the party playing the role of the verifier.) We will also restrict the number of rounds (associated to this verifier program) to be a polynomially bounded, polynomial time computable function.

Sometimes we wish to discuss probabilistic, polynomial time players who receive an additional "auxiliary" input (such an input may be, for example, a witness for the membership of the common input in some predetermined NP language). We may capture such situations by thinking of the auxiliary input as being incorporated in the interactive function (i.e. the party's interaction on common input x and auxilary y is captured by an oracle indexed by both x and y).

We will be interested in probabilistic machines which use interactive functions as oracles.

Definition 2.2 *Let $K(\cdot)$ be a probabilistic oracle machine, and A an interactive function. Then $K^{A_x}(x)$ is a random variable describing the output of K with oracle A_x and input x, the probability being over the random choices of K and A.*

The meaning of having A_x as an oracle is that K may specify a string η and, in one (special) step, obtain a random element from $A_x[\eta]$. We count the steps needed to specify η (and read the output), but the oracle invocation is just one step. It is understood that an invocation of the oracle on a string η returns a random element of $A_x[\eta]$, independently of any previous invocations of the oracle on other inputs.[3]

We call a function $f\colon \mathsf{N} \mapsto \mathsf{R}$ *negligible* if for all $c > 0$ and all sufficiently large n we have $f(n) < n^{-c}$. We call a function $f\colon \mathsf{N} \mapsto \mathsf{R}$ *non-negligible* if there exists $c > 0$ so that for all sufficiently large n we have $f(n) \geq n^{-c}$. We call $f\colon \{0,1\}^* \mapsto \mathsf{R}$ negligible if the function $n \mapsto \max_{x \in \{0,1\}^n} f(x)$ is negligible, and non-negligible if the function $n \mapsto \min_{x \in \{0,1\}^n} f(x)$ is non-negligible. As stressed above, non-negligible is not the negation of negligible but rather a very strong negation of it (and there exist functions which are neither negligible nor non-negligible).

3 A Definition of a Proof of Knowledge

Let $R \subseteq \{0,1\}^* \times \{0,1\}^*$ be a binary relation. Our aim is to define a "system of proofs of knowledge for R." For simplicity, we restrict our attention to polynomially bounded relations (and, unless otherwise stated, all relations in this paper are assumed to be such). Note that the most natural and important class of proofs of knowledge, namely those of "knowledge of a witness for an NP statement," correspond to the special case of NP relations.

The heart of the proof system is the verifier, which remains fixed for our entire discussion. This fixed verifier may interact with arbitrary provers, and we will relate the behavior of the verifier in these interactions with assertions concerning knowledge of the corresponding provers.

For the purpose of defining proofs of knowledge there is no need to restrict the verifier computationally, although in most applications one asks that it be probabilistic, polynomial time.

We make no assumptions concerning the possible provers (in contrast to previous formalizations). We don't even assume that they send messages that can be computed (say nothing about efficiently computed) from the information they receive (i.e., their initial input and in-coming messages). That is, provers are arbitrary interactive functions.

We wish to define the "knowledge of P about x which may be deduced from the interaction of P with V (on input x)". Clearly, this knowledge contains the transcript of the interaction. Yet, in case the interaction is accepting and this event is not incidental, one can say more on the knowledge of P. Namely, the ability of P to "often" lead the verifier to accept may say something about the knowledge of P. The crucial observation, originating in [14], is that the "knowledge of P about x (deduced by interaction)" can be captured by whatever can be *efficiently computed* on input x and access to the oracle P_x.

The phrase "efficiently computed on input x and access to an oracle P_x" is made precise in the definition of a "knowledge extractor." The straightforward approach is to require that the knowledge extractor is a probabilistic polynomial-time oracle machine. Indeed this is the approach taken in some previous works (if one translates their ideas to this slightly different setting). We will replace the strict requirement that the knowledge extractor works in polynomial-time by a more adaptive requirement which relates the running time of the knowledge extractor to the probability that the verifier is convinced. The advantages of this approach have already been discussed and will be further discussed below.

Let $p(x)$ be the probability that prover P convinces verifier V to accept on input x. In its simplest form, the requirement we impose is that the extractor succeed in outputting a witness in (expected) time proportional to $1/p(x)$. In actuality, we will introduce a "knowledge error function"

[3] A stricter alternative is obtained by fixing the prover's sequence of coin tosses and treating it as auxiliary input to the prover. Note that all known "proofs of knowledge" satisfy also this more strict requirement. The fact that the strict requirement implies the main one can be shown by techniques similar to those used in Appendix C.

$\kappa(\cdot)$ and ask that the extractor succeed in outputting a witness in (expected) time proportional to $1/(p(x)-\kappa(x))$. Intuitively, $\kappa(x)$ is the probability that the verifier might accept even if the prover did not in fact "know" a witness. We note that in applications $\kappa(x)$ is small, and often it is zero (cf. §4.4 and §5). The precise definition follows.

Definition 3.1 (System of proofs of knowledge) *Let R be a binary relation, and $\kappa\colon \{0,1\}^* \to [0,1]$. Let V be an interactive function which is computable in probabilistic, polynomial time. We say that a V is a* knowledge verifier for the relation R *with* knowledge error κ *if the following two conditions hold.*

- Non-triviality: *There exists an interactive function P^* so that for all $x \in L_R$, all possible interactions of V with P^* on common input x are accepting (i.e.* $\Pr[\mathrm{tr}_{P^*,V}(x) \in \mathtt{ACC}_V(x)] = 1$*).*

- Validity (with error κ): *There exists a constant $c > 0$ and a probabilistic oracle machine K such that for every interactive function P and every $x \in L_R$, machine K satisfies the following condition:*

  if $p(x) \stackrel{\mathrm{def}}{=} \Pr[\mathrm{tr}_{P,V}(x) \in \mathtt{ACC}_V(x)] > \kappa(x)$ then, on input x and access to oracle P_x, machine K outputs a string from the set $R(x)$ within an expected number of steps bounded by

  $$\frac{|x|^c}{p(x)-\kappa(x)}\,.$$

The oracle machine K is called a universal knowledge extractor*, and κ is called the* knowledge error function.

The next section is devoted to remarks on various features of this definition.

4 Remarks

We discuss various features of our definition, with particular regard to how it differs from previous definitions.

4.1 Provers which convince with non-negligible probability

Suppose the knowledge error is negligible. Clearly, if the verifier accepts with non-negligible probability then the knowledge extractor runs in average polynomial in $|x|$ time. This conclusion yields essentially what [6, 18] have considered as sufficient. Yet, as we have argued, this conclusion by itself does not suffice.

4.2 The efficiency of the provers and verifier

For the purpose of *defining* proofs of knowledge, there is no need to restrict the prover to polynomial-time. This is a point on which we disagree with previous works which claimed that it makes no sense to talk of the knowledge of unrestricted machines. Our definition is presented without assuming anything about the power of the prover, and it is a corollary that machines with no time bounds may know facts which cannot be deduced in (say) double exponential time (and so on). In particular, as we will see (cf. §7.2), it is meaningful, under our definition, to say that the prover in Shamir's interactive proof system for a PSPACE-complete language "knows" an accepting computation of a polynomial-space machine. One the other hand, provers which succeed in convincing a verifier of their knowledge can be reasonably efficient. For example, they may be implemented by polynomial-time programs. Furthermore, all "reasonable" interactive proofs for languages in NP (and in particular the zero-knowledge ones [12]) can be convinced by probabilistic polynomial-time provers which get an NP-witness as auxiliary input. (However, membership in an NP language can be proven via Shamir's result that IP = PSPACE. The corresponding prover is unlikely to be implementable in polynomial-time).

Note that we do not ask that the verifier be a probabilistic polynomial time interactive Turing machine, but just that it be an interactive function computable by one. This distinction is conceptually useful when we consider applications such as the graph non-isomorphism protocol [12] in which the verifier (of the proof of knowledge) is the prover of the graph non-isomorphism protocol, and thus not a probabilistic polynomial time interactive Turing machine. However, the part of this prover's program which implements the verifier (of the proof of knowledge) is indeed computable in probabilistic polynomial time.

4.3 The knowledge extractor

What should not *be given to the knowledge extractor.* We deviate from some previous works in that we define the knowledge of the prover only with respect to what is publicly available (i.e., the common input x, access to an oracle for the prover, and possibly the transcript). Some other works define the knowledge of the prover with respect to the auxiliary information available to the prover as well as its sequence of coin tosses (which may[4] not be known to the verifier). To justify our choice we remind the reader that the definition of "proof of knowledge" is supposed to capture the knowledge of the prover *demonstrated by the interaction* and not merely the knowledge of the prover. Hence, there seems to be little motivation and/or justification to talk about the knowledge of a machine with respect to something which is not known to the outside (i.e., verifier). In particular, only the common input (of the interaction) should be given as input to the knowledge extractor, and the auxiliary input or local coins of the prover should certainly not be given.

One thing that the knowledge extractor can do. In all examples we are aware of, the knowledge extractor proceeds by trying to find several (not more than polynomially many) related accepting transcripts. For example, the knowledge extractor presented in Appendix E tries to find a single accepting transcript in addition to the one given as input. Clearly such a knowledge extractor succeeds within an average number of steps which is inversely proportional to the density of the accepting transcripts (which is in other words the accepting probability). Note that if the proof of knowledge is zero-knowledge then a single accepting transcript (and in particular the one given as input) cannot suffice.

Universality of the knowledge extractor. In the above definition we require the existence of a universal knowledge extractor which works for all possible interactive functions P. Switching the quantifiers (i.e., requiring that for every interactive function P there exist a knowledge extractor K_P) would make little sense in practice since P in our conventions may depend on (non-uniform) auxiliary input of the "real" prover (cf. §2). However, the quantifiers may be switched if one considers only provers which are (uniform) interactive machines. For further discussion see the parenthetical subsection in [10, Sec. 4.1], which considers an analogous situation in the context of zero-knowledge. We stress that also in case the quantifiers are switched, the knowledge extractor (although it may depend on the prover) must be given *oracle access* to the prover. The reason

[4] Using the term "may" is indeed an understatement!

being that the prover's program may be highly inefficient (and therefore cannot be "incorporated" into the extractor).

4.4 The knowledge error function

The knowledge error function is a novelty of our definition.[5] Let us see why it is important.

Typically, "proofs of knowledge" are constructed by repeating an "atomic" protocol sufficiently many times. An atomic protocol for graph isomorphism, for example, is the following (cf. [12]).

Example. The input is a pair of (isomorphic) graphs G_1 and G_2. The prover generates a *single* random isomorphic copy of G_1 which we call H, and sends H to the verifier. The latter responds with a random query $i \in \{1, 2\}$. The prover replies to i by presenting an isomorphism between G_i and H. The verifier accepts if the permutation supplied by the prover is indeed an isomorphism between G_i and H.

Intuitively, this protocol does demonstrate some "knowledge" of an isomorphism between G_1 and G_2. Yet, previous definitions were unable to capture this fact; they were only able to show that sufficiently (i.e. super-logarithmic) many iterations of this protocol constituted a "proof of knowledge." This non-modular approach belies the basic intuition and is also not the natural approach to protocol design.

The introduction of the knowledge error function remedies these defects. In particular, we are able to capture "atomic" proofs of knowledge of the above type. Indeed, under our definition, the above is a proof of knowledge with knowledge error $1/2$. Furthermore, we are able to prove composition theorems which show how to reduce the knowledge error (cf. §5) and thus construct proofs of knowledge in a modular fashion.

Another motivation of the knowledge error function comes from cases where, for convenience, we have the verifier accept with some (usually small) probability even if the evidence supplied by the prover is not convincing. For example, we may do this to guarantee perfect completeness (i.e., the prover's ability to alway convince the verifier of valid statements). In such cases, the knowledge error can compensate for this small probability. The importance of this aspect of the knowledge error function, and the perfect completeness example, were pointed out to us by Feige (private communication, June 1992).

4.5 What about soundness?

We note that our definition makes no requirement for the case $x \notin L_R$. In particular, soundness (i.e., a bound on the prover's ability to lead the verifier to accept $x \notin L_R$) is not required. Consequently, a knowledge verifier for R does not necessarily define an interactive proof of membership in L_R. This is in contrast to previous definitions; they had the "validity" condition imply the soundness condition, so that the latter always held. We feel that our "decoupling" of soundness from validity is justified both conceptually and in the light of certain applications. Let us see why.

First, conceptually, it seems more natural to talk about extracting witnesses only when these witnesses exist. Furthermore, as long as one property is not known to imply the other it seems wrong to require the latter unless one really needs it.

Second, there are some natural applications (e.g., "zero-knowledge based" identification schemes) in which it is a-priori agreed that the protocol will be applied only to strings in some NP language (i.e., $x \in L_R \in \mathrm{NP}$). Such applications are better modeled by our definition than by previous ones.

[5] Although the ideas in [5] may be interpreted as pointing to a similar notion.

To be concrete, consider the following identification scheme based on the hardness of quadratic residuosity.

Example. A user A (Alice), who wishes to be able to securely remote-login to a mainframe computer (which we denote by V because it plays the role of verifier) chooses at random a pair of large primes and multiplies them to get a modulus N_A. She also chooses $Y_A \in Z_{N_A}^*$ at random, sets $X_A = Y_A^2 \bmod N_A$, and gives the pair (N_A, X_A) to V. All this is performed once in a life-time, when Alice is identified by other means. Later, whenever Alice wishes to remote-login, she sends her name (A) to V, who responds by sending the pair (N_A, X_A). She now provides a (zero-knowledge) proof that she "knows" a square root of $X_A \bmod N_A$. Besides the fact that A can provide the proof (completeness) we require that if Bob ($B \neq A$) were to attempt to remote-login as A then he (B) would fail. The point to note in (the formalization of) the latter requirement is that the interaction of B with V takes place on an input (namely (N_A, X_A)) which is in the underlying language L_R (the relation R here is $\{ ((N,X),Y) : Y^2 \equiv X \pmod{N} \}$ and the underlying language is $L_R = \{ (N,X) : X \text{ is a square } \bmod N \}$). So it suffices to require that the interaction of B with V on inputs *in this language* "proves possession of a witness." *What happens on interactions on input not in the language is immaterial to the security of the identification scheme.* Thus the requirements for a secure (zero-knowledge based) identification scheme are more faithfully modeled by our Definition 3.1 than by previous definitions (which required that *any* proof of knowledge of a relation R be an interactive proof of membership in L_R).

We stress that we are not, of course, saying that soundness is *always* redundant. Rather, the above discussion justifies our choice not to make soundness a part of the definition of a proof of knowledge. In cases where soundness is necessary, it can be viewed as a separate, additional property that the knowledge verifier must satisfy. Furthermore, it is possible that some applications call for other kinds of conditions on $x \notin L_R$. One possibility, which we call *strong validity*, is discussed in Appendix B.

4.6 Relaxing the non-triviality requirement

The prover guaranteed by the non-triviality requirement must convince the verifier in all interactions of $x \in L_R$. This requirement, met in all known protocols, is not essential to the definition of a proof of knowledge. In general one may require that the existence of a prover that convinces the verifier, on input x, with probability $C(x)$. As far as polynomial-time (or even more powerful) verifiers are concerned any choice of a polynomial-time constructible bound, $C(\cdot)$, which is both non-negligibly greater than $\kappa(\cdot)$ and bounded above by $1-2^{-\mathrm{poly}(\cdot)}$, is equivalent.[6] In fact, following the ideas in [9], one can eliminate the error probability in the completeness condition altogether and derive the definition as in the previous section. However, although the last transformation does preserve validity, it does not necessarily preserve the complexity of the prover and its zero-knowledge property.[7]

[6]When saying that these choices are equivalent, as long as the above requirements are satisfied, we mean that existence of a verifier which satisfies one permissible bound yields the existence of another verifier which satisfies the second bound. Furthermore, the complexity both of the verifier and of the prover (meeting the completeness condition) is preserved (and so are zero-knowledge properties).

[7]In this context we note, however, that the zero-knowledge too may be preserved, as long as one is willing to make a complexity assumption, by further applying the transformation of [2].

4.7 A word about computationally convincing proofs of knowledge

Some works (cf. [4, 5]) consider the situation in which the class of provers for which the protocol is supposed to be a "proof of knowledge" is restricted to the class of probabilistic, polynomial time interactive Turing machines with auxiliary input.[8] Typically, the protocols in question rely on the use of problems which are intractable for the prover(s). This is the case of *computationally convincing* (zero-knowledge) proofs, also known as *arguments* (cf. [3]).

Our definitions may be adapted to cover such settings as well. We would restrict the class of provers for which validity is required to hold to the class of interactive functions computable in probabilistic, polynomial time by interactive machines. We would, however, also relax slightly the validity requirement by asking that it only be true for sufficiently long inputs. More precisely, we would require that for each probabilistic, polynomial time computable interactive function P (prover) there exist a constant n_P such that for each $x \in L_R$ of length at least n_P, machine K satisfies the following condition:

> if $p(x) \stackrel{\text{def}}{=} \Pr[\mathrm{tr}_{P,V}(x) \in \mathtt{ACC}_V(x)] > \kappa(x)$ then, on input x and access to oracle P_x, machine K outputs a string from the set $R(x)$ within an expected number of steps bounded by $|x|^c/(p(x) - \kappa(x))$.

In applications, $\kappa(x)$ could be set to $1/\mathrm{poly}(x)$ for some specific $\mathrm{poly}(\cdot)$. Alternatively, following [7], one can use $\kappa(\cdot)$ as a shorthand for "smaller than any function of the form $1/\mathrm{poly}(\cdot)$". However, a much better alternative is to set $\kappa(\cdot)$ to be a specific negligible function (e.g., $\kappa(x) = 2^{-\sqrt[5]{|x|}}$) related to a specific intractability assumption concerning the computational problem on which the scheme is based (e.g., DLP is intractable with respect to algorithms which run in time $2^{\sqrt[5]{n}}$ on inputs of length n).

Some ideas on the subject of "computationally convincing proofs of knowledge" appear in the work of Brassard, Crépeau, Laplante and Léger [5]. Although they do not present definitions, it would appear these ideas bear many similarities to ours. We discuss their work in Appendix A.

The fact that some variations are needed to treat the case of "computationally convincing proofs of knowledge" has been pointed out to us by Feige (private communication, June 1992).

5 Reducing the knowledge error via repetitions

One of the reasons to introduce the knowledge error function is the theorems established here. We show that the knowledge error may be reduced by composition.

First we consider sequential composition. Here $m = m(x)$ independent copies of the original protocol are executed on input x, and the verifier accepts iff all copies are accepting (we stress that by "independent" we mean that the verifier acts in each of the copies independently of the others; of course we don't assume this about prospective provers). If κ was the knowledge error of the original protocol then the knowledge error the resulting protocol is essentially κ^m. The more precise statement follows.

Notational convention: by $\mathrm{poly}(\cdot)$ we mean any sufficiently large polynomial in the length of the input (string).

Required assumption: $y \in R(x)$ can be found (if such exists) in exponential-time (i.e., time $2^{\mathrm{poly}(|x|)}$). Finally, we assume of course that $m(x) \leq \mathrm{poly}(|x|)$.

[8] For simplicity we ignore the auxiliary inputs in this discussion. They can be treated as outlined in §2.

Theorem 5.1 *Suppose that V is a knowledge verifier for the relation R with error $\kappa(\cdot)$. Let V_m denote the program that, on input x, sequentially executes the program V, on input x, for $m(x)$ times. Then V_m is a knowledge verifier for the relation R with error $\kappa_m(\cdot) \stackrel{\text{def}}{=} (1+1/\text{poly}(\cdot))\cdot\kappa(\cdot)^{m(\cdot)}$.*

The proof is in Appendix C.1.

With respect to error reduction via parallel repetitions we were only able to prove a statement concerning a special class of knowledge verifiers (which nonetheless contains all known verifiers). For further discussion see Appendix C.2.

Finally, we observe that tiny knowledge error can be eliminated.

Proposition 5.2 *Suppose that an element in $R(x)$, if such exists, can be found in time at most $t(x)$, given only x as input. Suppose V is a knowledge verifier for R with knowledge error smaller than $\frac{1}{2\cdot t(x)}$. Then, V is a knowledge verifier for R with knowledge error 0.*

We omit the proof which uses methods similar to those used in Appendix B.

The resulting formulation (namely, knowledge error 0) is often the simplest way of thinking about proofs of knowledge: we are saying that the knowledge extractor succeeds in time $|x|^c/p(x)$, where $p(x)$ is as in Definition 3.1. Many proofs of knowledge (e.g., the one presented in Appendix E) are of this type.

6 An equivalent formulation of validity

Following is an equivalent formulation of the validity condition. The new formulation is inspired by (yet is quite different in many respects from) the definition in [7]. Let $p(x)$ be as in Definition 3.1. Instead of asking that the knowledge verifier always output $y \in R(x)$, we ask only that it output $y \in R(x)$ with a probability bounded below by $p(x) - \kappa(x)$, and otherwise output a special symbol, denoted $\perp$, indicating "failure to find $y \in R(x)$". However, whereas originally the extractor had expected time proportional to $1/(p(x) - \kappa(x))$, we now give it only expected polynomial time. More precisely, letting κ: $\{0,1\}^* \mapsto [0,1]$, we have the following.

- New validity (with error κ): We say that the verifier V satisfies *new validity with error κ* if there exists a probabilistic expected polynomial-time oracle machine K such that for every interactive function P and every $x \in L_R$ it is the case that $K^{P_x}(x) \in R(x) \cup \{\perp\}$ and
$$\Pr[K^{P_x}(x) \in R(x)] \geq \Pr[\text{tr}_{P,V}(x) \in \texttt{ACC}_V(x)] - \kappa(x) .$$

Proposition 6.1 *The new validity condition is equivalent to the one given in Definition 3.1.*

Here we give the proof for the case $\kappa(x) = 0$. The proof for the general case is more complex and is in Appendix D.

Suppose, first, that K is a knowledge extractor satisfying the new definition. We construct a knowledge extractor K' that, on input x repeatedly invokes K (on x) until $K(x) \neq \perp$. Clearly, K' always outputs a string in $R(x)$, halting in expected time $\text{poly}(x)/\Pr[K(x) \in R(x)]$, which is bounded above by $\text{poly}(x)/\Pr[\text{tr}_{P,V}(x) \in \texttt{ACC}_V(x)]$. Hence, K' satisfies the condition in Definition 3.1. Suppose, now, that K is a knowledge extractor satisfying Definition 3.1. We construct a knowledge extractor K' that, on input x first generates a random transcript (i.e., $\text{tr}_{P,V}(x)$) and activates $K(x)$ if this transcript is accepting (i.e., in $\texttt{ACC}_V(x)$). Otherwise, K' halts immediately outputting $\perp$. One can easily verify that K' runs in expected polynomial-time and outputs $y \in R(x)$ with probability exactly $\Pr[\text{tr}_{P,V}(x) \in \texttt{ACC}_V(x)]$.

7 Applications

Our formalization, as well as that of [7], do suffice to prove the security of those schemes for encryption secure against chosen-cyphertext attack which rely on zero-knowledge proofs of knowledge (cf. §1.2). However, we prefer to describe here two applications to which our definition of "proof of knowledge" can be applied, whereas all the previous formalizations fail. The first application is a modular description of the zero-knowledge proof for Graph Non-Isomorphism (of [12]) which uses a "proof of knowledge of an isomorphism" as a subprotocol. The second application is to Shamir's interactive proof for PSPACE.

7.1 Zero-Knowledge proof of Graph Non-Isomorphism

The second author first realized the inadequacy of previous formulations of "proofs of knowledge" when Leonid Levin insisted that the zero-knowledge interactive proof for Graph Non-Isomorphism (of [12]) should be presented in a modular manner.[9] As many people noticed, the intuition behind this zero-knowledge proof is that the verifier first proves to the prover that it "knows" an isomorphism between one of the input graphs and the query graph that it presents to the prover.[10] If the prover is convinced then it answers the query by indicating to which of the two input graphs the query graph is isomorphic. By doing so the prover yields no knowledge to the verifier, since the verifier "knows" to which of the two input graphs the query is isomorphic, yet the prover's answer supplies statistical evidence that the two input graphs are not isomorphic. This intuitive idea, taken from the Quadratic Non-Residousity zero-knowledge proof of [14], has indeed guided the development of the zero-knowledge proof system for GNI, but plays no part in the formal description and proof of correctness appearing in [12] (and [14]). Levin complained, rightfully, against this inelegant and non-modular approach. The second author's answer, at the time, was that an elegant proof which uses the subprotocol and its properties in a modular fashion is not possible due to lack of appropriate definitions.[11]

One definition that was lacking at the time was that of the information hiding property of the subprotocol used to prove "possession of knowledge". Specifically, that subprotocol, which consists of the parallel version of the zero-knowledge proof of Graph Isomorphism, is not known to be zero-knowledge (and in light of [11] it is unlikely that a proof that it is zero-knowledge can ever be given). Nevertheless, this subprotocol is "witness indistinguishable" (in the sense defined latter by Feige and Shamir [7]) and this property suffices to the soundness of the interactive proof of GNI. However this entire issue is irrelevant to the current paper.

The other definition that was lacking at that time was an *adequate* definition of a proof of knowledge. An adequate definition of a "proof of knowledge" is needed to ensure that if the GNI-prover is convinced that the GNI-verifier "knows" an isomorphism between the query graph and one of the input graphs then indicating to which input graph the query graph is isomorphic yields no knowledge to the GNI-verifier.[12] To this end, the simulator (constructed to meet the zero-knowledge clause) uses the knowledge extractor guaranteed by the definition of a "proof of knowledge". However, as pointed out above, previous definitions of "proof of knowledge" are useless in the case the GNI-prover is not convinced with non-negligible probability. It follows that

[9] For sake of self-containment, this protocol is presented in Appendix E

[10] The prover in the zero-knowledge proof for GNI is the verifier in a "proof of knowledge of an isomorphism between two graphs"; whereas the verifier in the zero-knowledge proof for GNI is the party claiming and proving knowledge of an NP-witness for GI.

[11] It should be stressed that a proof of correctness of (the zero-knowledge property of) the protocol of does appear in [12]. The criticism points to the fact that the proof of correctness in [12] does not reflect the intuition just outlined.

[12] The reader may find it useful at this point to consult Appendix E.

the simulator will fail to construct the interactions in these cases which may occur with probability that is neither non-negligible nor negligible (see §1.2). In particular, consider the situation where for every $c > 0$ there exists an infinite sequence of inputs to the protocol such that on input of length n the GNI-prover is convinced with probability n^{-c}.

On the other hand, one can show that the subprotocol "for proof of knowledge of isomorphism" (presented in [12] and Appendix E) constitutes a (sound) proof of knowledge, according to the definitions presented in §3. It follows that the running time of the knowledge extractor is inversely proportional to the probability that the GNI-prover is convinced. Hence, the simulator for the GNI-protocol will run in expected polynomial-time and produce a perfect simulation of the interaction. Furthermore, it can be easily shown that the GNI-prover *while playing the role of the GI-verifier in the proof of knowledge* yields no knowledge to the GNI-verifier (since its messages are generated in probabilistic polynomial-time from its inputs).

7.2 What does the prover of a PSPACE language know?

Using our definition, it is possible to say that the verifier in Shamir's interactive proof for a PSPACE-complete language L is a knowledge verifier for the relation R_L consisting of pairs (x, c) where c is the middle configuration in the computation of a fixed machine accepting $x \in L$. Hence, one can say that (in some meaningful sense) any prover which convinces this verifier (with, say, probability 1) on input x, does know an accepting computation on input x.

Let us show how a knowledge extractor may find the middle configuration. For the rest of this subsection, we assume that the reader is very familiar with the interactive proof for QBF as presented in [17, Section 5]. The standard reduction of a PSPACE language to QBF associates the middle configuration in an accepting poly-space computation with the first block of t existential quantifiers in the formula. So in the rest of this subsection we will consider only the problem of retrieving a sequence of truth-values so that assigning these values to the above mentioned variables yields value `true` for the resulting formula.

First, we consider a straightforward method for retrieving these t boolean values. This method does work in case the prover convinces the verifier with probability 1 (but will have to be modified to deal with arbitrary provers). First the knowledge extractor asks the oracle for the first message of the prover which is a pair (N, v_0), where N is a large prime and v_0 is a non-zero residue mod N (the value of the arithmetic expression mod N). Next, the knowledge extractor proceeds in t rounds. In the i^{th} round, the extractor feeds the oracle the sequence $r_1, ..., r_{i-1} \in Z_N$ and gets the polynomial, p_i, which corresponds to the opening of the i^{th} variable, when the previous $i-1$ variables are set to $r_1, ..., r_{i-1}$, respectively. The extractor then finds a $\mu_i \in \{0, 1\}$ so that $p_i(\mu_i)$ is not equal to zero modulo N (such μ_i must exist since $\sum_{\mu \in \{0,1\}} p_i(\mu) \equiv v_{i-1} \not\equiv 0 \pmod{N}$). Round i is completed by setting $r_i = \mu_i$ and $v_i = p_i(r_i)$.

In general the above method may fail as it relies too heavily on the answers of the prover on boolean r_i's. An alternative approach is to select the r_i's uniformly in Z_N. The problem is that the resulting residual arithmetic expression no longer reflects the truth value of the residual boolean formula. To solve the problem we need to find the polynomial resulting by setting the r_i's to μ_i's by examining the polynomials which result by random settings of the r_i's. To see how this can be done, we need to take a closer look at the formula used by Shamir and its arithmetization. It can be seen that the polynomial p_i received from the prover in round i has coefficients which are polynomials in r_1 through r_{i-1}. Denote by $c_{i,j}(r_1, ..., r_{i-1})$ the polynomial in r_1 through r_{i-1} representing the j^{th} coefficient of p_i. The $c_{i,j}$'s are polynomials each of total degree at most $2(i-1) < 2t-1$, and we are interested in the values of $c_{i,j}(\sigma_1, ..., \sigma_{i-1})$. Using the ideas of [1] these values can be found via "interpolation" at $2t$ uniformly selected (yet dependent) points. Finally, we note that the knowledge extractor can tell whether it is given the correct polynomial at a point by carrying on the rest of the interactive proof using the oracle to the function P_x. Further details are omitted.

Acknowledgements

The second author thanks Leonid Levin for his interest in "proofs of knowledge" and his insistence that they have to be formalized in a sufficiently robust manner so that they can be used in applications such as the Graph Non-Isomorphism protocol.

We are grateful to Uri Feige for valuable criticisms of an earlier version of this paper. Specific credit to Feige's suggestions is given in the relevant places of the current manuscript.

References

[1] D. Beaver, and J. Feigenbaum, "Hiding Instances in Multioracle Queries," *Proc. of the 7th STACS*, 1990, pp. 37-48.

[2] M. Bellare, S. Micali and R. Ostrovsky, "The True Complexity of Statistical Zero-Knowledge," *Proceedings of the 22nd Annual ACM Symposium on the Theory of Computing,* ACM (1990), pp. 494-502.

[3] G. Brassard, D. Chaum, and C. Crépeau, "Minimum Disclosure Proofs of knowledge," *JCSS*, Vol. 37, No. 2, 1988, pp. 156–189.

[4] J. Boyar, C. Lund and R. Peralta, "On the Communication Complexity of Zero-Knowledge Proofs." 1989.

[5] G. Brassard, C. Crépeau, S. Laplante and C. Léger, "Computationally Convincing Proofs of Knowledge," *Proc. of the 8th STACS*, 1991.

[6] U. Feige, A. Fiat, and A. Shamir, "Zero-Knowledge Proofs of Identity", *Journal of Cryptology*, Vol. 1, 1988, pp. 77-94.

[7] U. Feige, and A. Shamir, "Witness Indistinguishability and Witness Hiding Protocols," *Proceedings of the 22nd Annual ACM Symposium on the Theory of Computing,* ACM (1990), pp 416-426.

[8] Z. Galil, S. Haber, and M. Yung, "Symmetric Public-Key Encryption", *Advances in Cryptology - Crypto85 proceedings*, Lecture Notes in Computer Science, Vol. 218, Springer-Verlag, 1986, pp. 128-137.

[9] M. Furer, O. Goldreich, Y. Mansour, M. Sipser, and S. Zachos, "On Completeness and Soundness in Interactive Proof Systems", *Advances in Computing Research: a research annual*, Vol. 5 (S. Micali, ed.), pp. 429-442, 1989.

[10] O. Goldreich, "A Uniform-Complexity Treatment of Encryption and Zero-Knowledge", *J. of Cryptology*, to appear.

[11] O. Goldreich, and H. Krawczyk, "On Sequential and Parallel Composition of Zero-Knowledge Protocols", *17th ICALP*, Lecture Notes in Computer Science, Vol. 443, Springer-Verlag, 1990, pp. 268–282.

[12] O. Goldreich, S. Micali, and A. Wigderson, "Proofs that Yields Nothing but Their Validity or All Languages in NP Have Zero-Knowledge Proof Systems", *JACM*, Vol. 38, No. 1, July 1991.

[13] O. Goldreich, and Y. Oren, "Definitions and Properties of Zero-Knowledge Proof Systems", TR-610, Computer Science Dept., Technion, Haifa, Israel. Submitted to *Jour. of Cryptology.*

[14] S. Goldwasser, S. Micali, and C. Rackoff, "The Knowledge Complexity of Interactive Proof Systems", *SIAM J. on Computing*, Vol. 18, No. 1, 1989, pp. 186-208.

[15] S. Haber, "Multi-Party Cryptographic Computations: Techniques and Applications", PhD Dissertation, Computer Science Dept., Columbia University, Nov. 1987.

[16] Y. Oren, "On the Cunning Power of Cheating Verifiers: Some Observations about Zero-Knowledge Proofs," *Proceedings of the 28th Annual IEEE Symposium on the Foundations of Computer Science,* IEEE (1987), pp. 462-471.

[17] A. Shamir, "IP=PSPACE," *Proceedings of the 31st Annual IEEE Symposium on the Foundations of Computer Science,* IEEE (1990), pp. 11-15.

[18] M. Tompa and H. Woll, "Random Self-Reducibility and Zero-Knowledge Interactive Proofs of Possession of Information," University of California (San Diego) Computer Science and Engineering Dept. Technical Report Number CS92-244 (June 1992). (Preliminary version in *Proceedings of the 28th Annual IEEE Symposium on the Foundations of Computer Science,* IEEE (1987), pp. 472-482.)

A Previous Definitions of Proofs of Knowledge

For sake of self-containment we review below the definitions of "proof of knowledge" appearing in the literature. In general there are two generally cited formulations appearing in [6] and in [18]. In addition, there is the better (but lesser known) formulation of Feige and Shamir [7]. Finally, there is work on "computationally convincing proofs of knowledge" [4, 5].

"Proof of Knowledge" according to Feige, Fiat and Shamir [6] The definition presented in [6] refers only to parties which work in probabilistic polynomial-time, yet may have auxiliary input (which is not necessarily generated efficiently). The knowledge extractor is given the prover's program and auxiliary input and may run the prover's program as a subroutine (yet being charged for the time).[13] The knowledge extractor is required to produce good output only for provers and inputs for which the prover has a non-negligible probability of convincing the verifier on that input. Specifically, it is required that

> for every constant $a > 0$ there exists a probabilistic polynomial-time extractor M so that for all constants $b > 0$, all provers P, and all sufficiently large x, r, k, if $\Pr[(P,V)(x,r,k) = \texttt{ACC}] > |x|^{-a}$ then $\Pr[M(\mathbf{desc}(P), x, r, k) \in R(x)] > 1 - |x|^{-b}$. ($\mathbf{desc}(P)$ denotes the description of P).

The string k in the above definition denotes a-priori knowledge of P (given in the form of auxiliary input) where r denotes the prover's sequence of coin tosses. The fact that k is given to the knowledge extractor, though being indeed conceptually disturbing, can be justified in several applications (and in particular those in [6]). We stress that the definition of [6] does not guarantee one knowledge extractor which works regardless of the prover's success probability but rather a sequence of extractors each relevant for a different "measure" of non-negligence. As claimed in the our text this is conceptually unsatisfactory and inadequate for many applications in which a proof of knowledge is used as a subroutine. It should be said that "proofs of knowledge" are not used as subprotocols in [6], but rather as the "thing itself" (and hence our critic of their definition is only weakly relevant, if at all, to the results of that paper).

"Proof of Knowledge" according to Tompa and Woll [18] The definition presented in [18] differs slightly from the one of [6]. It allows the verifier to run for an arbitrary (not necessarily polynomial) amount of time. The running time of the knowledge extractor is polynomial in the length of the input and in the running time of the verifier. As explained in §4.3, we don't believe that this choice is justified. The knowledge extractor in the [18] definition is given as input the *prover's view* of the interaction with the verifier, which contains among other things the prover's auxiliary input (denoted k in the definition of [6] presented above). The requirement concerning the output of the verifier is that the event "on input x the verifier is convinced yet the knowledge extractor fails to find $y \in R(x)$" happens very rarely (i.e. with probability smaller than ϵ for some $\epsilon < 1$). The probability is taken over the random coin tosses of both parties (for any fixed input x and fixed auxiliary input k). Clearly, this definition suffers from all the disadvantages of the definition of [6] discussed above. Furthermore, if ϵ is indeed fixed, as suggested by the definition in [18], then protocols satisfying their definition are useless even in a stronger sense: the prover may convince the verifier with probability $\epsilon/2$ and yet the knowledge extractor is required nothing.

[13]The extractor may try to analyze the prover's program by other means but Feige, Fiat and Shamir claim that this does not make sense. In any case the knowledge extractors that they present only use the prover's program as a "black-box".

Tompa and Woll were indeed aware of this point and seem to suggest to eliminate the problem by applying the protocol iteratively sufficiently many times. This is indeed a good suggestion. However, several problems remain. First a conceptual problem: their Lemma 3 (hereafter referred to as the *Composition Lemma*) indeed offers a useful tool, but it does not provide a general satisfactory definition of a "proof of knowledge". More annoying is the fact that the Composition Lemma constructs better protocols via *sequential* composition of worse ones. It is not clear (and furthermore it seems unlikely) that a parallel composition will have the same affect. Finally, the Composition Lemma is applicable only to relations R which are in BPP.

"Proof of Knowledge" according to Feige and Shamir [7] The definition presented in [7] looks similar to the one in [6], but in fact it is fundamentally different. The critical point is that the definition in [7] treats potential provers uniformly with respect to the probability they lead the verifier to accept. In this sense, the definition in [7] is similar to our definition. Specifically, the knowledge extractor, denoted M, runs in expected polynomial-time (rather than in strict polynomial-time as in [6]) and outputs an element of $R(x)$ with probability that is at most non-negligibly smaller than the probability that the verifier accepts on input x. Specifically, it is required that

> there exists a probabilistic *expected* polynomial-time extractor M so that for all constants $b > 0$, all provers P, and all sufficiently large x, r, k,
>
> $$\Pr[(P, V)(x, r, k) = \mathtt{ACC}] > \Pr[M(\mathbf{desc}(P), x, r, k) \in R(x)] - |x|^{-b}$$

Consequently this definition does not suffer from the main criticism raised against the definition of [6]. However, it still suffers from the other problems such as the fact that k is given to M. Furthermore, it does not capture "knowledge" of super-polynomial-time provers.

Work on "computationally convincing proofs of knowledge". Brassard, Crépeau, Laplante and Léger [5] study "computationally convincing proofs of knowledge" (the "validity" condition refers only to probabilistic, polynomial-time provers). They do not present formal definitions so we found it difficult to compare their work to ours, but the ideas appear to have some relation. They too propose an "adaptive" requirement linking the running time of the extractor to the success of the prover. Specifically, they appear to consider a particular class of protocols, namely those consisting of k rounds, each of which contains a "challenge" (from verifier to prover) which the prover may correctly answer with probability 1/2 if he correctly "guesses" a coin toss of the verifier. They require that the extractor succeed in time linear in $1/\varphi$, where $2^{-k} + \varphi$ is the "probability of undetected cheating." The quantity in quotes was not defined precisely, particularly for the case of the input being in the language, but if $2^{-k} + \varphi$ is interpreted as the probability that the verifier accepts, then it is like our definition with the knowledge error set to 2^{-k}.

Brassard et. al. [5] also raise some criticisms of the definitions of [6, 18], but their criticism is the opposite of ours: whereas we suggest that the previous definitions are too weak (and propose a stronger definition) they suggest that the previous definitions are already too strong.

B Soundness and Strong Validity

For completeness, we state here also the standard soundness condition (for interactive proof systems). We remind the reader that we view soundness as an additional property that a knowledge verifier may (or may not) satisfy.

Definition B.1 (Additional possible properties of a system of proofs of knowledge) *Let R be a binary relation, and suppose that V is a knowledge verifier for the relation R with knowledge error κ. We define two additional properties that V may satisfy:*

- soundness: *For every interactive function P, and for all $x \notin L_R$, most of the possible interactions of V with P on common input x are rejecting (i.e., $\Pr[\mathrm{tr}_{P,V}(x) \in \mathtt{ACC}_V(x)] < 1/2$).*
- strong validity (with error κ): *Let K be the universal knowledge extractor, and $c > 0$ be the constant guaranteed by the validity condition of Definition 3.1. Then, for every interactive function P and every $x \notin L_R$, machine K satisfies the following condition:*

  if $p(x) \stackrel{\text{def}}{=} \Pr[\mathrm{tr}_{P,V}(x) \in \mathtt{ACC}_V(x)] > \kappa(x)$ then, on input x and access to oracle P_x, machine K outputs the special symbol $\perp$ within an expected number of steps bounded by
  $$\frac{|x|^c}{p(x) - \kappa(x)}$$

As usual, the completeness (or non-triviality) and soundness conditions merely state that there is a gap between the probability that a prover may convince the verifier on $x \in L_R$ (which by the completeness condition is exactly 1) and the probability that a prover may convince the verifier on $x \notin L_R$ (which by the soundness condition is at most 1/2). Validity (resp., strong validity) is a more refined condition regarding the behavior of arbitrary provers on $x \in L_R$ (resp., arbitrary strings). Specifically, validity relates the probability that the prover convinces the verifier on $x \in L_R$ and the average time it takes the knowledge extractor to find a $y \in R(x)$ in the case $x \in L_R$. Strong validity is an analogous requirement regarding $x \notin L_R$. Validity, soundness, and strong validity are not always independent. Namely,

Proposition B.2 *Validity and soundness imply strong validity for NP relations.*

The proof that follows is for the case $\kappa = 0$.

Recall that an NP relation is a polynomially bounded relation $R(\cdot,\cdot)$ which is decidable in polynomial time. Suppose an NP relation R possesses a knowledge verifier which (in addition) satisfies the soundness condition. Without loss of generality[14], we may assume the error probability in the soundness condition is at most $2^{-p(n)}$, where $p(\cdot)$ is a polynomial bounding the length of witnesses as a function of the length of the input. Let K be the universal knowledge extractor (satisfying the validity condition). Fix a deterministic procedure, with running-time $2^{p(n)} \cdot \mathrm{poly}(n)$, for deciding L_R (e.g., the one which scans through all possible witnesses for the given input).

We construct a new knowledge extractor, denoted K', for the above proof of knowledge, satisfying also strong validity. On input x and oracle access to P_x, machine K' runs in parallel the extractor K (with input x and oracle P_x) and the decision procedure for L_R, fixed above. Suppose K halts before the decision procedure terminates, and yields an output y. Machine K' checks whether $R(x,y)$ is true (it can do this in polynomial time) and if so outputs y; otherwise it outputs $\perp$. On the other hand, suppose the decision procedure halts while K is still running. If the decision is negative ($x \notin L_R$) then K' outputs $\perp$; else it continues to run K to whatever outcome this might yield.

We note that the running time of K' is (within a polynomial factor of) that of K when $x \in L_R$, and at most (within a polynomial factor of) $2^{p(|x|)}$ otherwise. But in the latter case, the probability $p(x) = \Pr[\mathrm{tr}_{P,V}(x) \in \mathtt{ACC}_V(x)]$ is at most $2^{-p(|x|)}$, so that the running time of K' is expected

[14]The error probability in the soundness condition may be reduced, as usual, by repetitions.

$|x|^{O(1)}/p(x)$ in both cases. The fact that K' is a knowledge extractor for R which satisfies (validity and) strong validity follows.

Finally, we note that the above transformation preserves (upto polynomial factors) the running time of the knowledge verifier, and, as long as we do the error-reduction in a suitable way (for example, by serial composition), it also preserves zero-knowledge.

C Reducing the Knowledge Error via Repetitions

We prove the claims of §5. Let us first recall the notation and assumptions introduced there. By $\mathrm{poly}(\cdot)$ we mean any sufficiently large polynomial in the length of the input (string). By assumption the messages of the verifier can be computed in polynomial-time, and $y \in R(x)$ can be found (if such exists) in exponential-time (i.e., time $2^{\mathrm{poly}(x)}$). Consequently, failure of the knowledge extractor occurring with exponentially small probability (i.e., probability $2^{-\mathrm{poly}(x)}$) can be ignored. Finally, we assume of course that $m(x) \leq \mathrm{poly}(x)$.

C.1 Reducing the Knowledge Error via Sequential Composition

Suppose that V is a knowledge verifier with error $\kappa(\cdot)$ for the relation R, and let K be a knowledge extractor witnessing this fact. Let V_m denote the program that, on input x, sequentially executes the program V, on input x, for $m(x)$ times. Theorem 5.1 asserts that V_m is a knowledge verifier with error $\kappa_m(\cdot) \stackrel{\mathrm{def}}{=} (1+1/\mathrm{poly}(\cdot)) \cdot \kappa(\cdot)^{m(\cdot)}$ for the relation R. The theorem is proven by constructing a knowledge extractor, denoted K_m, as described below.

Suppose that P_m is a prover which, on input x, leads V_m to accept with probability $p_m(x) > \kappa_m(x)$. Loosely speaking, we observe that there exists an i, $0 \leq i \leq m(x)-1$, and a partial transcript of i iterations so that, relative to this partial transcript, the $i+1^{\mathrm{st}}$ iteration is accepting with probability at least $\sqrt[m(x)]{p_m(x)}$. The idea is to use the guaranteed knowledge extractor, K, on the $i+1^{\mathrm{st}}$ iteration of V_m, relative to an appropriate partial i-iteration transcript. Details follow.

For simplicity, we assume here that all transcripts are equally likely. Let T_i denote the set of all possible partial transcripts of the first i iterations, and $A_i \subseteq T_i$ denote the set of partial (i-iteration) transcripts in which all the i iterations are accepting. Let $a_i \stackrel{\mathrm{def}}{=} |A_i|/|T_i|$ ($a_0 \stackrel{\mathrm{def}}{=} 1$). For every $\alpha \in A_i$, let $q(\alpha)$ denote the accepting probability of the $i+1^{\mathrm{st}}$ iteration relative to a partial transcript α, and c_{i+1} denote the average of $q(\alpha)$ taken over all $\alpha \in A_i$.

The following sequence of claims lead to the construction of the knowledge extractor K_m.

Claim 1: for every i, $0 \leq i < m(x)$, it holds that $a_{i+1} = a_i \cdot c_{i+1}$.

Proof: Clearly,

$$\begin{aligned}
|A_{i+1}| &= \sum_{\alpha \in A_i} \frac{|T_{i+1}|}{|T_i|} \cdot q(\alpha) \\
&= |T_{i+1}| \cdot \frac{|A_i|}{|T_i|} \cdot \frac{\sum_{\alpha \in A_i} q(\alpha)}{|A_i|}
\end{aligned}$$

and the claim follows. □

Claim 2: there exists an i, $0 \leq i < m(x)$, such that

1. $c_{i+1} \geq \sqrt[m(x)]{p_m(x)}$.
2. $a_i \cdot (c_{i+1} - \kappa(x)) > \frac{p_m(x)}{\mathrm{poly}(x)}$.

Proof: By Claim 1, $p_m(x) = \prod_{i=1}^{m(x)} c_i$, and Part (1) follows. Using $p_m(x) > \kappa_m(x)$, we get

$$\begin{aligned} c_{i+1} &\geq \sqrt[m(x)]{1 + 1/\text{poly}(x)} \cdot \kappa(x) \\ &= \left(1 + \frac{1}{\text{poly}(x)}\right) \cdot \kappa(x) \end{aligned}$$

and hence $c_{i+1} - \kappa(x) \geq c_{i+1}/\text{poly}(x)$. Using $a_i \cdot c_{i+1} \geq p_m(x)$, Part (2) follows. $\Box$

Notation: Let i be as guaranteed by Claim 2, and denote $\delta_{i+1} \stackrel{\text{def}}{=} c_{i+1} - \kappa(x)$. Let $A_{i,t}$ denote the set of partial transcripts in A_i containing only partial transcripts relative to which the $i+1^{\text{st}}$ iteration accepts with probability bounded below by $\kappa(x) + 2^t \delta_{i+1}/\text{poly}(x)$ and above by $\kappa(x) + 2^{t+1}\delta_{i+1}/\text{poly}(x)$, where $\text{poly}(\cdot)$ is a specific polynomial which depends on $m(\cdot)$ and the time required to find $y \in R(x)$. Namely,

$$A_{i,t} \stackrel{\text{def}}{=} \left\{\alpha \in A_i \ : \ \kappa(x) + 2^t \cdot \frac{\delta_{i+1}}{\text{poly}(x)} \ \leq \ q(\alpha) \ < \ \kappa(x) + 2^{t+1} \cdot \frac{\delta_{i+1}}{\text{poly}(x)} \right\}$$

Claim 3: Let i and $A_{i,t}$ be as above. Then there exists an t, $1 \leq t < \text{poly}(x)$, such that $|A_{i,t}| \geq 2^{-t} \cdot |A_i|$.

Proof: Assume, on the contrary, that the current claim does not hold. Then

$$\begin{aligned} c_{i+1} &< \kappa(x) + \frac{\delta_{i+1}}{\text{poly}(x)} + \sum_{t \geq 1} \frac{|A_{i,t}|}{|A_i|} \cdot \left(2^{t+1} \cdot \frac{\delta_{i+1}}{\text{poly}(x)}\right) \\ &< \kappa(x) + \frac{\delta_{i+1}}{\text{poly}(x)} + \sum_{t=1}^{\text{poly}(x)} 2^{-t} \cdot \left(2^{t+1} \cdot \frac{\delta_{i+1}}{\text{poly}(x)}\right) \\ &< \kappa(x) + \delta_{i+1} \\ &= c_{i+1} \end{aligned}$$

and contradiction follows. $\Box$

Claim 4: There exists an i, $0 \leq i < m(x)$, and an j, $1 \leq j < \text{poly}(x)$, such that at least a 2^{-j} fraction of the $\alpha \in T_i$ satisfy

$$q(\alpha) > \kappa(x) + 2^j \cdot \frac{p_m(x)}{\text{poly}(x)}$$

Proof: Let i as guaranteed by Claim 2. Rephrasing Claim 3, we get that there exists an t, $1 \leq t < \text{poly}(x)$, such that at least a $2^{-t} \cdot a_i$ fraction of the $\alpha \in T_i$ satisfy $q(\alpha) > \kappa(x) + 2^t \cdot \delta_{i+1}/\text{poly}(x)$. Substituting $j = t + \log_2(1/a_i)$ and using Part (2) of Claim 2, the claim follows. $\Box$

Using Claim 4, we are now ready to present the knowledge extractor K_m. Machine K_m runs in parallel $m(x) \cdot \text{poly}(x)$ copies of the following procedure, each with a different pair (i,j), $1 \leq i \leq m(x)$ and $1 \leq j \leq \text{poly}(x)$. By saying "run several copies in parallel" we mean execute these copies so that t steps are executed in each copy before step $t+1$ is executed in any other copy[15].

The copy running with the pair (i,j), generates $M \stackrel{\text{def}}{=} 2^j \cdot \text{poly}(x)$ random partial transcripts of i-iterations, denoted $\gamma_1, ..., \gamma_M$, and runs M copies of the knowledge extractor K in parallel, each using a corresponding partial transcript (γ_k). The sub-procedure, indexed by the triple (i,j,k), uses

[15] Actually, the condition can be related. For example, it suffices to require that at least t steps are executed in each copy before step $2 \cdot t$ is executed in any other copy.

the partial transcript γ_k to convert queries of the basic knowledge extractor (i.e., K) into queries concerning the $i+1^{\text{st}}$ iteration. Namely, when K is invoked it asks queries to an oracle describing the messages of a prover interacting with V. However, K_m has access to an oracle describing prover P_m (which is supposedly interacting with V_m). Hence, K_m needs to simulate an oracle describing a basic prover (interacting with V), by using an oracle describing P_m. This is done by prefixing each query of K with the i-iteration partial transcript γ_k generated above.

To analyze the performance of K_m consider the copy of the procedure running with a pair (i,j) satisfying the conditions of Claim 4. If this is the case, then with very high probability (i.e., exponentially close to 1) at least one of the partial transcripts generated by this copy has the property that, relative to it, the $i+1^{\text{st}}$ iteration accepts with probability at least $\kappa(x) + 2^j p_m(x)/\text{poly}(x)$. It follows that the corresponding copy of the sub-procedure will halt, outputting $y \in R(x)$, within $\frac{\text{poly}(x)}{2^j \cdot p_m(x)}$ steps (on the average). Since the $(i,j)^{\text{th}}$ copy of the procedure consists of $2^j \cdot \text{poly}(x)$ copies of the sub-procedure running in parallel, this copy of the procedure will halt in expected time $\frac{\text{poly}(x)}{p_m(x)} < \frac{\text{poly}(x)}{p_m(x)-\kappa_m(x)}$. The entire knowledge extractor consists of polynomially many copies of the procedure, running in parallel, and hence it also runs in expected $\frac{\text{poly}(x)}{p_m(x)-\kappa_m(x)}$ time as required.

Remark: We believe that V_m is a knowledge verifier with error $\kappa(\cdot)^{m(\cdot)}$ for the relation R (rather than just being a knowledge verifier with error $(1+1/\text{poly}(\cdot)) \cdot \kappa(\cdot)^{m(\cdot)}$ for this relation). The difference is of little practical importance, yet we consider the question to be of theoretical interest.

C.2 Reducing the Knowledge Error via Parallel Composition

A fundamental problem with presenting a parallel analogue of the above argument is that we cannot fix a partial transcript for the other iterations while working with one selected iteration (which was possible and crucial to the proof used in the sequential case). Furthermore, even analyzing the profile of accepting transcripts is more complex.

As before, let $p_m(x)$ denote the accepting probability, here abbreviated by $p(x)$, and let $\delta(x) \stackrel{\text{def}}{=} p(x) - \kappa_m(x)$. Consider a $m(x)$-dimensional table in which the dimensions correspond to the $m \stackrel{\text{def}}{=} m(x)$ parallel executions, where the $(r_1, ..., r_m)$-entry in the table corresponds to the transcript when the verifier uses coin tosses r_1 in the first execution, r_2 in the second execution, and so on. Since a $p(x)$ fraction of the entries are accepting transcripts, it follows that there exists a dimension i so that at least a $\sqrt[m(x)]{p(x)-\delta(x)/2}$ fraction of the rows in the i^{th} dimension contain at least $\delta(x)/2m(x)$ accepting entries. Furthermore, there exists a j, $0 \le j \le \log_2(\text{poly}(x)/\delta_m(x))$, so that at least a $2^j \cdot \sqrt[m(x)]{p(x)-\delta(x)/2}$ fraction of the rows in the i^{th} dimension contain at least $\frac{p(x)-\delta(x)/2}{2^j \text{poly}(x) \cdot \sqrt[m(x)]{p(x)-\delta(x)/2}}$ accepting entries.

Getting back to the problem of using the knowledge extractor K (of the basic verifier V), we note that we need to simulate an oracle to K using an oracle describing P_m. The idea used in the sequential case is to augment all queries to the P-oracle by the same partial transcript. However, we can no longer guarantee high accepting probability for one execution relative to a fix transcript of the other (parallel) executions.

We can however treat the special case in which the basic knowledge extractor, K, operates by generating random transcripts and keeping a new transcript only if it satisfies some polynomial-time predicate with respect to the transcripts kept so far. Details omitted. We remark that the known knowledge extractors do operate in such a manner.

D Equivalence of Two Formulations of Validity with Error

We now prove the equivalence of the definitions of validity with error given in Definition 3.1 and in §6, respectively. We assume that whenever $\Pr[\mathrm{tr}_{P,V}(x)\in \mathtt{ACC}_V(x)] > \kappa(x)$, we have $\Pr[\mathrm{tr}_{P,V}(x)\in \mathtt{ACC}_V(x)] > \kappa(x) + 2^{-\mathrm{poly}(x)}$ as well. Alternatively, we may assume that there exist an exponential time algorithm for solving the relation R (i.e., finding $y \in R(x)$ if such exists within $2^{\mathrm{poly}(x)}$ steps). The proof extends the argument presented in §6, for the special case $\kappa = 0$, yet in one direction an additional idea is required.

Let us start with the easy direction. Suppose that a verifier V satisfies validity with knowledge error $\kappa(\cdot)$ by the definition in §6. Let K be a knowledge extractor satisfying this definition. We construct a knowledge extractor K' that, on input x repeatedly invokes K (on x) until $K(x) \neq \perp$. Clearly, K' always outputs a string in $R(x)$, halting in expected time $\mathrm{poly}(x)/\Pr[K(x) \in R(x)]$ which is bounded above by $\mathrm{poly}(x)/(\Pr[\mathrm{tr}_{P^*,V}(x) \in \mathtt{ACC}_V(x)] - \kappa(x))$. Hence, K' satisfies the condition in Definition 3.1.

Suppose that a verifier V satisfies validity with knowledge error $\kappa(\cdot)$ by Definition 3.1, and let K be a knowledge extractor witnessing this fact. Let $c > 0$ be the constant satisfying the condition on the running-time of K. Namely, that its expected running-time is bounded above by $|x|^c/(\Pr[\mathrm{tr}_{P,V}(x)\in \mathtt{ACC}_V(x)] - \kappa(x))$. Assume, without loss of generality, that with very high probability (i.e., exponentially close to 1) K halts within at most $2^{\mathrm{poly}(x)}$ steps[16]. We construct a knowledge extractor K' that, on input x runs $K(x)$ with the following modification. Machine K' proceeds in iterations, starting with $i = 1$, and terminating after at most $\mathrm{poly}(x)$ iterations. In iteration i, machine K' executes $K(x)$ with time bound $2^i \cdot |x|^c$. If K halts with some output y then K' outputs y and halts. Otherwise (i.e., K' does not halt within $2^i \cdot |x|^c$ steps), machine K' halts with probability $\frac{1}{2}$ with output $\perp$ and otherwise proceeds to iteration $i + 1$. We stress that in all iterations, K uses the same internal coin tosses. In fact, we can record the configuration at the end of iteration i and consequently save half of the time spent in iteration $i + 1$. Clearly, the expected running-time of $K'(x)$ is bounded above by

$$\sum_{i=1}^{\mathrm{poly}(x)} \frac{1}{2^{i-1}} \cdot (2^i \cdot |x|^c) = \mathrm{poly}(x)$$

We now evaluate the probability that, on input x, machine K' outputs $y \in R(x)$. It is guaranteed that, on input x, the extractor K outputs $y \in R(x)$ within $T(x) \leq |x|^c/(\Pr[\mathrm{tr}_{P,V}(x)\in \mathtt{ACC}_V(x)] - \kappa(x))$ steps on the average (and by hypothesis $T(x) < 2^{\mathrm{poly}(x)}$). Hence, with probability at least $\frac{1}{2}$, on input x, machine K outputs $y \in R(x)$ within $2 \cdot T(x)$ steps. The probability that K' conducts $2 \cdot T(x)$ steps (i.e., K' reaches iteration $\log_2(T(x)/|x|^c)$) is $|x|^c/T(x) \geq \Pr[\mathrm{tr}_{P,V}(x)\in \mathtt{ACC}_V(x)] - \kappa(x)$. Hence, K' satisfies the condition in §6.

E The Zero-Knowledge proof of Graph Non-Isomorphism

Following is the basic ingredient of the zero-knowledge proof for Graph Non-Isomorphism (GNI) presented in [12].

Common input: Two graphs G_1 and G_2 of n vertices each.
Objective: In case the graphs are not isomorphic, supply (statistical) evidence to that affect.

[16] This can be achieved by running the exponential time solver in parallel to K. Alternatively, assuming that if $\Pr[\mathrm{tr}_{P,V}(x) \in \mathtt{ACC}_V(x)] > \kappa(x)$ then $\Pr[\mathrm{tr}_{P,V}(x) \in \mathtt{ACC}_V(x)] > \kappa(x) + 2^{-\mathrm{poly}(x)}$, we can implement a probabilistic exponential-time solver using K.

Step V1: The *GNI-verifier* selects uniformly $i \in \{1,2\}$, and a random isomorphic copy of G_i, hereafter denoted H and called the *query*, and sends H to the *GNI-prover*. Namely, H is obtained by selecting a random permutation π, over the vertex-set, and letting the edge-set of H consist of pairs $(\pi(u), \pi(v))$ for every pair (u, v) in the edge-set of G_i.

Step VP: The GNI-verifier "convinces" the GNI-prover that he (i.e., the GNI-verifier) "knows" an isomorphism between H and one of the input graphs. To this end the two parties execute a witness indistinguishable proof of knowledge (with zero error) for graph isomorphism. (Such a protocol is described below.) In that proof of knowledge the GNI-verifier acts as the prover while the GNI-prover acts as the verifier.

Step P1: If the GNI-prover is convinced by the proof given at step VP, then he finds j such that H is isomorphic to G_j, and sends j to the GNI-verifier. (If H is isomorphic to neither graphs or to both the GNI-prover sets $j = 1$; this choice is arbitrary.)

Step V2: If j (received in step P1) equals i (chosen in step V1) then the GNI-verifier accepts, else he rejects.

It is easy to see that if the input graphs are not isomorphic then there exists a GNI-prover which always convinces the GNI-verifier. This meets the completeness condition of interactive proofs. To show that some sort of soundness is achieved we use the witness indistinguishability of the subprotocol used in Step VP. Loosely speaking, it follows that no information about i is revealed to the GNI-prover and therefore if the input graphs are isomorphic then the GNI-verifier rejects with probability at least one half (no matter what the prover does).[17]

The demonstration that the GNI-prover is zero-knowledge is the place where the notion of proof of knowledge plays a central role. As required by the zero-knowledge condition we have to construct, for every efficient program playing the role of the GNI-verifier, an efficient simulator which outputs a distribution equal to that of the interaction of the verifier program with the GNI-prover. Following is a description of such a simulator. The simulator starts by invoking the verifier's program and obtaining a query graph, H, and a transcript of the execution of step VP (this is obtained when the simulator plays the role of the GNI-prover which is the knowledge-verifier in this subprotocol). If the transcript is not accepting then the simulator halts and outputs it (thus perfectly simulating the real interaction). However, if the transcript is accepting the simulator must proceed (otherwise its output will not be correctly distributed). The simulator needs now to simulate step P1, but, unlike the real GNI-prover, the simulator does not "know" to which graph H is isomorphic. The key observation is that the simulator can obtain this information (i.e., the isomorphism) from the knowledge extractor guaranteed for the proof of knowledge (taking place in step VP), and once the isomorphism is found producing the rest of the interaction (i.e., the bit j) is obvious. Using our definition (of proof of knowledge with zero error), the simulator can find the isomorphism in expected $\text{poly}(n)/p(G_1, G_2, H)$ time, where $p(G_1, G_2, H)$ is the probability that the GNI-prover is convinced by the proof of knowledge in step VP. Since this module in the simulator is invoked only with probability $p(G_1, G_2, H)$, the simulator runs in expected polynomial-time, and the zero-knowledge property follows. *We stress that carrying out this plan is not possible when using any of the previous definitions of "proof of knowledge".*

To complete the description of the above protocol we present a (witness indistinguishable) proof of knowledge of Graph Isomorphism. This proof of knowledge can be easily adapted to a proof of knowledge of an isomorphism between the first input graph and one of the other two input graphs.

[17] Reducing the cheating probability further can be done by iterating the above protocol either sequentially or in parallel. However, this is not our concern here.

Common input: Two graphs H and G of n vertices each.
Objective: In case the graphs are isomorphic, the GI-prover has to "prove knowledge of ψ", where ψ is an isomorphism between H and G.

Note: In our application the GNI-verifier plays the role of the GI-prover, while the GNI-prover plays the role of the GI-verifier.
Notation: Let $t \stackrel{\rm def}{=} t(n) \stackrel{\rm def}{=} n^2$.

Step p1: The *GI-prover* selects uniformly t random isomorphic copies of H, denoted $K_1, ..., K_t$ and called the *mediators*, and sends these graphs to the *GI-verifier*. Namely, K_i is obtained by selecting a random permutation π_i over the vertex-set, and letting the edge-set of K_i consist of pairs $(\pi_i(u), \pi_i(v))$ for every pair (u, v) in the edge-set of H.

Step v1: The GI-verifier selects uniformly a subset S of $\{1, 2, ..., t\}$ and sends S to the GI-prover.

Step p2: For every $i \in S$, the *GI-prover* sets $a_i = \pi_i$, where π_i is the permutation selected in step p1 to form K_i. For every $i \in \{1, ..., t\} - S$, the *GI-prover* sets $a_i = \pi_i\psi$, where π_i is as before, ψ is the isomorphism between G and H (known to the GI-prover), and $\pi\psi$ denotes composition of permutations (or isomorphisms). The GI-prover sends $a_1, a_2, ..., a_t$ to the GI-verifier.

Step v2: The GI-verifier checks if, for every $i \in S$, the permutation a_i (supplied in step p2) is indeed an isomorphism between the graphs H and K_i. In addition, the GI-verifier checks if, for every $i \in \{1, 2, ..., t\} - S$, the permutation a_i (supplied in step p2) is indeed an isomorphism between the graphs G and K_i. If both conditions are satisfied (i.e., all t permutations are indeed what they are supposed to be) then the GI-verifier accepts (i.e., is convinced that the GI-prover knows an isomorphism between G and H).

One can show that the above GI-verifier constitutes a knowledge-verifier (with zero error) for Graph Isomorphism. This is done by considering all possible choices of $S \subseteq \{1, 2, ..., t\}$ for a fixed set of mediators $K_1, ..., K_t$. Denote by s the number of subsets S for which the GI-verifier accepts. A knowledge extractor, given one accepting interaction (i.e., containing a good S) tries to find another one (i.e. a good subset different from S). Having two good subsets clearly yields an isomorphism between G and H (i.e., using any index in the symmetric difference between the good subsets). Clearly, if $s = 1$ then there exists no good subset other than S. In this case the extractor finds an isomorphism by exhaustive search (which is always performed in parallel to the attempts of the extractor to find a different good subset). The exhaustive search requires less than 2^t steps, but dominates the total running time only in case $s = 1$ (in which case the accepting probability is $1/2^t$). Yet, for any $s > 1$, the expected number of tries required to find a different good subset is

$$\frac{1}{(s-1)/(2^t-1)} < \frac{2^t}{s-1} \leq \frac{2 \cdot 2^t}{s}$$

(the last inequality follows from $s \geq 2$). Since $s/2^t$ is the probability that the GI-verifier accepts, the extractor described above indeed runs in expected time inversely proportional to the accepting probability of the GI-verifier. Our claim follows.