A Stronger Notion of Proofs of Knowledge

Shai Halevi* Silvio Micali
MIT - LCS
545 Technology Square
Cambridge, MA 02139, USA

November 12, 1996

Abstract

Proofs of knowledge are central to cryptographic protocols, and many definitions for such proofs have been proposed. In this work we put forward a new definition, strictly stronger than previous ones, and argue that it captures better the intuition of what proofs of knowledge should be. We show that our definition implies some new and desirable properties.

1 Introduction

Proofs of knowledge play a crucial role in cryptographic protocols, and there have been several attempts to "zoom in" on an adequate formalization of this notion. On a very high level, a proof of knowledge is a protocol by which one party ("the Prover") proves to another party ("the Verifier") that it "knows a secret" about a given common input. Clearly, one such way would be exhibiting the secret in question, However, the notion becomes interesting (and hard to formalize properly!) when the secret is not directly revealed.

The intuitive notion of a *proof of knowledge* first appeared in the conference version of Goldwasser, Micali and Rackoff's paper [GMR85] (as a side remark on their notion of a zero-knowledge proof), as well as in a paper of Fisher, Micali, and Rackoff [FMR84], but was not formally defined in either of these papers.

The first attempts at a formal definition [FS90, FFS87, TW87] did not address several possible scenarios[1] These weaknesses were observed and corrected in a more recent work of Bellare and Goldreich [BG92]. However, as we argue below, even the later definition falls short of capturing the intuitive meaning of a proof of knowledge, and suffers from some undesirable side-effects.

It is thus our goal in this paper to provide a stronger and meaningful notion of a proof of knowledge. Notice that, although the interesting proofs of knowledge are the zero-knowledge ones, defining zero-knowledge proofs of knowledge is not much of a challenge once an appropriate definition of proofs of knowledge has been found. Accordingly, in this extended abstract we solely focus on defining proofs of knowledge.

THE COMMON CORE OF ALL PRIOR NOTIONS. For the sake of simplicity, in the intuitive discussion below we denote the common input to the prover and verifier by x, and its secret by y. According to all prior notions, saying that a prover P proves (by means of an interactive protocol) to a verifier V that it knows the secret y, means that P "could explicitly compute y with minimal extra effort." In all previous definitions (and in ours too) this minimal extra effort is formulated in terms of a polynomial-time Turing machine (called the *knowledge extractor*) that, after interacting with P a few times, outputs the secret y. (It is instructive to think of the

*Contach author at MIT - LCS, NE43-342, 545 Tech. Square, Cambridge, MA 02139, USA. E-mail: shaih@theory.lcs.mit.edu

[1] Most notably, the case where the verifier is convinced with probability which is not non-negligible, and the case where the prover is not restricted to probabilistic-polynomial-time.

knowledge extractor is as the "sub-conscience" of the prover. Thus, if the prover convinces the verifier that it knows y, then it can easily reconstruct y by itself.)

Notice, however, that proofs of knowledge are often zero-knowledge ones. (Informally, this means that no polynomial-time machine can compute the secret by interacting with the prover.) This seemingly contradicts the existence of a knowledge extractor. The reason that no contradiction actually occurs is that all prior notions assume that the prover is *resettable*; namely, it is possible to restore the entire state of the prover at the beginning of its computation. (The restored state includes the contents of the prover's random tape, should it be a probabilistic Turing machine.)

SEMANTIC DIFFICULTIES OF PRIOR NOTIONS. Let us now point out what we believe to be the major conceptual difficulty in all prior approaches. Assume that a prover P interacts with a Verifier V on a common input x, and convinces V that it knows a secret y about x. Then it ought to be the case that P's knowledge of y is demonstrated in V's view of this interaction. As this view is actually a *probability distribution* which it determined by both P and V (and their probabilistic choices), it must be *this* distribution that demonstrates P's knowledge of y.

However, prior definitions allow a knowledge extractor, K, to outputs y after interacting several times with P in any way it wants. It is thus possible that the distribution which enables K to compute y is *totally unrelated* to the one that P and V generate together. In a sense, therefore, the ability of K to compute y stems from the "wrong reasons", and has nothing to do with the fact that V is convinced.

1.1 Our Notion of Proofs Of Knowledge

We too make use of knowledge extractors in our definition. However, to bridge the "semantic gap" discussed above, we restrict the way a knowledge extractor operates. We illustrate the rationale behind these restrictions by breaking them into three separate, though dependent, components.

1. THE VERIFIER-VIEW CONSTRAINT. Prior definitions allow a knowledge extractor to interact with the prover P in an unrestricted way, without any relation to the way in which the verifier V interacts with P. Since we insist that V's view is what demonstrates P's knowledge of the secret y, we force our knowledge extractor K to compute a matching view of V for each interaction with P. That is, not only should K send to P messages that V might have sent, but it should also generate a sequence of coin tosses that would cause V to send those messages.

Syntactically, for each interaction with P, K produces a private output consisting of sequence of "coin tosses," and sends to P exactly those messages that V would have send given these coin tosses and the prior message received from P.

2. THE SAME-DISTRIBUTION CONSTRAINT. Notice that the verifier-view constraint says nothing about the distribution of the views generated by K. We thus strengthen this constraint by demanding that, for each interaction of K with P, the view of V which is generated by K has exactly the same distribution as V's view in a random P-V interaction. While insisting that each P-K interaction is distributed just like a random P-V interaction, we allows these interactions to be *dependent*.

3. THE ACCEPTING-VIEW CONSTRAINT. So far, we only specified how a knowledge extractor K should interact with the prover. Now we specify the manner in which K should compute the secret y from these interactions.

As we said above, K generates verifier views while interacting with the prover P, so at the end of this communication it can use these views to extract the secret y. Although in general, some of these views correspond to accepting conversations and other correspond to rejecting conversations, we insist in our definition that only the *accepting views* are used to compute y. Indeed, if the fact that V accepts an interaction with P has any meaning, the interactions accepted by V should be the ones that demonstrate P's knowledge of the secret y. Thus, it should be possible to compute y based solely on V's accepting views.

To express this constraint, we split our knowledge extractor into two components: an *interacting component* (which interacts with P and generates V's views as described so far) and a *non-interactive component*, a function that computes y from just the accepting views generated by the first component. By splitting K this way, we also ensure that K's ability of computing y does not depend on K's internal state. (As K's internal state might also encode V's rejecting views.) Notice, that sieving the accepting views from the rejecting ones is easy because one can use a view to simulate V and check whether it accepts or rejects.

We call a knowledge extractor that obeys all three constraints above a *conservative knowledge-extractor*, and the resulting notion a *strong proof of knowledge*.

1.1.1 Properties of Our Notion

PROVER FEASIBILITY. The same-distribution constraint of our definition, apart from being crucial in its own right, also implies other desirable properties. One such property is discussed next.

In most proofs of knowledge, the secret y can be computed from the input x by an exponential-time exhaustive search. Therefore, a proof that the prover P knows y is meaningless if the time that the extractor K and P together take to compute y is exponential in $|x|$. Although all prior definitions demand that K be polynomial-time, they all ignore the time that P may invest when interacting with K. But if, in order to compute y, the extractor must "trick the prover" into performing an exhaustive search for y (using queries that the original verifier would never use), then, in what sense does the original prover-verifier conversation demonstrate that the prover knows the secret y ? The above discussion suggests that the knowledge extractor in a proof of knowledge should have an additional property which we term *prover feasibility*. Informally, this property requires that

> *While extracting the secret, the total running-time of both the knowledge extractor and the prover should be comparable to that of the prover when interacting with the verifier.*

We formally define prover feasibility in Section 5, and prove that it is satisfied by our stronger notion of proof of knowledge. In Section 6 we also show that prior notions only satisfy a weaker prover-feasibility property.

MODEL INDEPENDENCE AND MODEL DEPENDENCE. We believe that the right notion of proof of knowledge should not depend on the particular computational model of prover. The right notion of a proof of knowledge should remain valid when prover is a probabilistic Turing machine, an infinite table, a quantum computer, a person, or any other interactive process.

By contrast, in our definition we demand that a verifier be probabilistic polynomial-time Turing machine. Part of this choice is just a matter of convenience, and part is not. Indeed, we wish that a proof of knowledge system specifies a *constructive* and *efficient* method to verify whether a prover has really knowledge of a given secret. Certainly, a probabilistic polynomial-time Turing machine is such a method. It is possible (though cumbersome) to generalize our notion to abstract models of computations for the verifier, as long as the notion of the verifier-view is well defined. Since we consider the verifier-view a central component in the semantics of a proof of knowledge, we doubt that any definition which survives in a model where the verifier view is not well define, can actually capture the meaning of this notion.

As for the knowledge extractor, in this extended abstract we assume that it too is a probabilistic Turing machines. Just like in the case of the verifier, the knowledge extractor too can be carried to other models, as long as it can efficiently generate verifier views.

1.1.2 Exemplifying Our Notion

It is often the case in cryptography that good protocols predate the definitions which explain why they are good. Protocols for proofs of knowledge belong to this category. Our definition can be viewed as distilling the true virtue of these good protocols from their algorithmic description. To illustrate this point, we show that "famous protocols" from the literature are examples of our new, stronger notion. In particular, in Section 4 we show

that the quadratic residuosity protocol of [GMR85] is one such example. It thus appears that prior works have claimed and proved less about their protocols than was both possible and desirable.

1.2 Organization

The rest of the paper is organized as follows: In Section 2 we introduce a few notations. In Section 3 we present our definition of a *strong proof-of-knowledge*, and in Section 4 we exemplify this definition by showing a well-known protocol which satisfies it. In Section 5 we define the notion of *prover feasibility* and prove that our definition enjoys that property. In Section 6 we show that prior definitions possess a weaker notion of prover feasibility. Finally, in Section 7 we discuss alternative ways to formulate our constraints and explain why these other alternatives are too restrictive.

2 Notations

In this section we introduce a few notations which we use throughout the paper. Since the syntax required to describe protocols may become quite awkward, we choose in this extended abstract to use somewhat informal syntax rather than an exact one. Also, for simplicity, all definitions of this extended abstract are presented assuming that both the prover and the verifier are Turing machines.

2.1 Interactive Turing Machines.

We rely on the intuitive notion of interactive Turing-machines as machines which can send messages to one another. The reader is referred to [GMR85] for a more formal definition. A pair of interactive Turing machine (M_1, M_2) defines a *protocol* in the usual way. Four aspects of a protocol are of particular interest to us: transcript, view, output, and running-time.

TRANSCRIPT. Let M_1, M_2 be interactive Turing machines; let (M_1, M_2) be the corresponding protocol and let (a, b) be an input to this protocol (so that a is a private input to M_1, and b is a private input to M_2). Consider an execution of (M_1, M_2) on input (a, b); then, the sequence of messages exchanged between M_1 and M_2 in this execution is called *the transcript* of the execution. We denote by $\mathbf{trans}\{M_1(a) \leftrightarrow M_2(b)\}$ the distribution over these transcripts (which is ultimately induced by the coin-tosses of M_1 and M_2).

VIEW. By $\mathbf{view}_{M_2}\{M_1(a) \leftrightarrow M_2(b)\}$ we denote the distribution over the pairs $(\mathcal{T}, \mathcal{C})$, where $\mathcal{T}$ and $\mathcal{C}$ are, respectively, the transcript and M_2's coin tosses in a random execution of protocol (M_1, M_2) on input (a, b).

OUTPUT. Although in a computation of (M_1, M_2) on input (a, b), each of M_1, M_2 may produce a private output, we denote by $\mathbf{out}\{M_1(a) \leftrightarrow M_2(b)\}$ the distribution (over the coin-tosses of M_1, M_2) on the output of M_2 only. (This is so because in the sequel we are only interested in the output of the second machine, the verifier.)

RUNNING-TIME. The distribution of the running time of M_1 over the executions of the protocol (M_1, M_2) on input (a, b) is denoted $\mathbf{time}_{M_1}\{M_1(a) \leftrightarrow M_2(b)\}$. The distribution of the running time of M_2 is denoted $\mathbf{time}_{M_2}\{M_1(a) \leftrightarrow M_2(b)\}$.

2.2 Experimenting Machines

To develop our notion, we consider "sequences of computations" of a special interactive Turing-machine E with an "ordinary" interactive Turing-machine M on a given input (a, b). In each of these computations, M starts in its starting configuration (including the state of the control as well as the contents of all the work tapes and the random tape), but E retains its state from one computation to the next. Such machine E is called an *experimenting machine*.

A little more formally, an experimenting machine E has a special state called the *reset state*. When involved in a protocol with a usual Interactive Turing Machine M, the semantics is that whenever E enters the reset state, the computational state of M is reset to the initial configuration. For syntactical reasons, we require that E also produces a private output at the end of each interaction just before entering the reset state. (We use this private output in the sequel to enable the knowledge-extractor to print out the coin-tosses of the verifier.)

If E is an experimenting machine, we refer to the sequence of interactions between E and M as an *experiment* and denote it by $E(a)^{M(b)}$. This notation was chosen to stress the similarity between an experimenting machine and the equivalent notion of an oracle machine (used in other definitions). Next we define the transcript, running-time and result of an experiment.

EXPERIMENT TRANSCRIPT. Let E be an Experimenting Machine and M be an Interactive Turing Machine. Then, a computation of experiment $E(a)^{M(b)}$ consists of a sequence of protocol executions, each with its own transcript and private E-output. The vector consisting of all these (transcript,E-output) pairs is called *experiment transcript* (or e-transcript for short). We denote the distribution over these e-transcripts (induced by the random tapes of E and M) by $\mathbf{etrans}\{E(a)^{M(b)}\}$.

For an integer i, the distribution over the i'th entry in the e-transcript vector is denoted by $\mathbf{etrans}\{E(a)^{M(b)}\}_i$. (If the vector contains less than i entries, then its ith entry is defined to be the special symbol $\perp$.)

RUNNING-TIME. In a random execution of $E(a)^{M(b)}$, we denote the distribution of the running times of E by $\mathbf{etime}_E\{E(a)^{M(b)}\}$ and the distribution over the running times of M by $\mathbf{etime}_M\{E(x)^{M(b)}\}$. We also denote by $\mathbf{etime}_E\{E(a)^{M(b)}\}_i$ (resp. $\mathbf{etime}_M\{E(a)^{M(b)}\}_i$) the distribution over the running times of E (resp. M) while generating the i'th entry in the e-transcript. If an e-transcript contains less than i entries then both running times are defined to be 0. Therefore $\mathbf{etime}_E\{E(a)^{M(b)}\} = \sum_i \mathbf{etime}_E\{E(a)^{M(b)}\}_i$ (and similarly for M).

RESULT. The result of an execution is the "public output" of the experimenting machine at the end of the experiment (it should not be confused with the private output after each interaction). We denote the distribution over the result by $\mathbf{result}\{E(a)^{M(b)}\}$.

3 Strong Proofs of Knowledge

In this section we describe our definition of strong proof of knowledge for a binary relation. In what follows, let $R \subset \Sigma^* \times \Sigma^*$ be a binary relation such that for all $(x,y) \in R$ we have $|y| \leq Q(|x|)$. for some fixed polynomial Q. For every string x we denote the set of strings which are R-related to x by $R(x)$. Namely, $R(x) \stackrel{\text{def}}{=} \{y : (x,y) \in R\}$.

We consider a proof of knowledge system to be a protocol (P,V), where P is the honest prover, and V the specified verifier. We require that, if the canonical way of proving knowledge (i.e., P) is followed, then one can convince V all the time. There is, however, a probability that a cheating prover may convince V without actually having any knowledge about the common input x. We refer to this probability as the error probability of the proof of knowledge system, and denote it by ε. More generally, a (possibly cheating) prover P' may have some probability p of convincing V. As long as $p > \varepsilon$, our definition requires that it is possible to extract the knowledge from such a P' with probability (polynomially) close to $p - \varepsilon$.

Definition 1: Let ε be a real number, $0 \leq \varepsilon \leq 1$. A *Strong Proof of Knowledge* with error ε for a binary relation R, consists of a pair of Interactive Turing-machines (P,V), (where V always outputs either YES or NO), with the following properties:

Completeness. For all $(x,y) \in R$, $\Pr[\mathbf{out}\{P(x,y) \leftrightarrow V(x)\} = YES] = 1$

Validity with error ε. There exists a probabilistic, strict polynomial-time Experimenting Machine K, a polynomial-time computable function EXTRACT, a constant $c > 0$, and a function $\#(\cdot)$, so that for every $x \in \Sigma^*$ and every Interactive Turing machine P'

0. All the e-transcripts having positive probability in $\mathbf{etrans}\{K(x)^{P'}\}$ have exactly $\#(x)$ pairs $(\mathcal{T},\mathcal{C})$ in them. (This requirement is not really necessary, and is introduced solely for the purpose of simplifying the syntax and notations in the sequel.)

1 & 2. For $i = 1,\ldots,\#(x)$, $\mathbf{etrans}\{K(x)^{P'}\}_i = \mathbf{view}_V\{P' \leftrightarrow V(x)\}$.

(i.e., the ith entry in the e-transcript is a random V-view)

3. Let $\mathbf{accept}\{K(x)^{P'}\}$ be the random variable obtained by substituting the special symbol $\perp$ for each rejecting view in $\mathbf{etrans}\{K(x)^{P'}\}$. We require that $\text{EXTRACT}(\mathbf{accept}\{K(x)^{P'}\})$ be in $R(x) \cup \{\perp\}$, and $\Pr\left[\text{EXTRACT}(\mathbf{accept}\{K(x)^{P'}\}) \in R(x)\right] \geq (\Pr[\mathbf{out}\{P' \leftrightarrow V(x)\} = YES] - \varepsilon)^c$

We call a pair (K,EXTRACT) as above a *conservative knowledge extractor* for the strong proof of knowledge (P,V).

REMARK. The above definition can be slightly strengthened by requiring that P, V and K be uniform and polynomial in $\log(1/\varepsilon)$. That is, each of P, V, K in the definition above is given another input 1^k, and we require that the error be at most 2^{-k}. We chose the above weaker formulation to be consistent with previous definitions in the literature.

4 Strong Proofs of Knowledge Exist

The purpose of this section is to show that the new, stronger notion of proof of knowledge is *realizable*. In fact, we believe that most (if not all) of the proof of knowledge discussed in the literature actually satisfy our stronger requirements. To prove our point, we chose discuss the first (zero-knowledge) proof of knowledge. This is a protocol of Goldwasser, Micali, and Rackoff [GMR85] where a Prover convinces (in zero knowledge) a verifier that it knows a square root (mod n) of a give square (mod n). Essentially the same protocol appears in [FMR84].

THE GOLDWASSER-MICALI-RACKOFF PROTOCOL. Casting the Goldwasser-Micali-Rackoff protocol in our syntax, we have a binary relation $R = \{(\langle n,x\rangle, y) : x,y \in Z_n^* \text{ and } x \equiv y^2 \bmod n\}$. The Goldwasser-Micali-Rackoff protocol (with error $1/2$) is as follows

	Prover $P(n,x,y)$	transcript	Verifier $V(n,x)$
1.	Pick $r \in_R Z_n^*$ Compute $s = xr^2 \bmod n$	$s \rightarrow$	
2.		$\longleftarrow \sigma$	Pick a random bit $\sigma \in \{0,1\}$
3.	Compute $t = yr \bmod n$ if $\sigma = 1$ and $t = r$ otherwise	$t \rightarrow$	
4.			Check that $t^2 \equiv s \bmod n$ if $\sigma = 1$ and $xt^2 \equiv s \bmod n$ otherwise

Theorem 1: The Goldwasser-Micali-Rackoff Protocol is a strong proof of knowledge with error $1/2$.

Proof: The completeness property is obvious. For the Validity property, let P' be any prover and denote the probability of an accepting $P' \leftrightarrow V(n,x)$ conversation by $p_{n,x}$. The Experimenting machine just samples two $P' \leftrightarrow V(n,x)$ conversations. (Thus, the number of pairs in the vector $\mathbf{etrans}\{K(n,x)^{P'}\}$ is always two.) If

σ and σ' (the random bits which are sent by V in, the first and second conversations respectively) are not the same, and if both conversations are accepting, then y can be efficiently computed from t and t'. Thus we have

$$\begin{aligned}&\Pr[\textsc{Extract}(\mathbf{accept}\{K(n,x)^{P'}\}) = \sqrt{x} \bmod n] \\ &= \Pr[\text{both conversations are accepting and } \sigma \neq \sigma'] \;\geq\; p_{n,x} \cdot (p_{n,x} - 1/2) \;>\; (p_{n,x} - 1/2)^2\end{aligned}$$

□

5 Strong Proofs of Knowledge Are Prover-Feasible

In this section we show that strong proofs of knowledge possess the desired prover-feasibility property. Recall that, informally, prover feasibility entails that, when communicating with the knowledge-extractor, the prover spends about the same time as it does when communicating with the verifier. This is defined formally below

Definition 2: Let V be an interactive Turing machines and let K be an Experimenting machine. We say that K *maintains the feasibility of interacting with* V if there exists a polynomial $Q(\cdot)$ such that for every input x and every Interactive Turing machine P it holds that

$$E\left[\mathbf{etime}_P\{K(x)^P\}\right] \leq Q(|x|) \cdot E\left[\mathbf{time}_P\{P \leftrightarrow V(x)\}\right]$$

where the expectation is taken over the random coins of K, V and P.

Theorem 2: Let (P, V) be a strong proof of knowledge system for the binary relation R, and let $(K, \textsc{Extract})$ be a conservative knowledge extractor for (P, V). Then, K maintains the feasibility of interacting with V.

Proof: For any input $x \in \Sigma$ and any Interactive Turing Machine P', let $c_{x,P'}$ denote the number of entries in $\mathbf{etrans}\{K(x)^{P'}\}$. Then, because K runs in (strict) polynomial time, there exists a positive polynomial $Q(\cdot)$ such that, for all x and P', $c_{x,P'} \leq Q(|x|)$. Moreover, by definition of the running time of an experiment

$$E\left[\mathbf{etime}_{P'}\{K(x)^{P'}\}\right] = E\left[\sum_i \mathbf{etime}_{P'}\{K(x)^{P'}\}_i\right] = \sum_i E\left[\mathbf{etime}_{P'}\{K(x)^{P'}\}_i\right]$$

but since the distribution of the transcript (the first entry) in every pair in $\mathbf{etrans}\{K(x)^{P'}\}$ is identical to the distribution $\mathbf{trans}\{P' \leftrightarrow V(x)\}$, then for every integer i we have

$$E\left[\mathbf{etime}_{P'}\{K(x)^{P'}\}_i\right] = \begin{cases} E\left[\mathbf{time}_{P'}\{P' \leftrightarrow V(x)\}\right] & \text{if } i \leq c_{x,P'} \\ 0 & \text{otherwise} \end{cases}$$

Therefore we have

$$E\left[\mathbf{etime}_{P'}\{K(x)^{P'}\}\right] = \sum_{i=1}^{c_{x,P'}} E\left[\mathbf{time}_{P'}\{P' \leftrightarrow V(x)\}\right] \leq Q(|x|) \cdot E\left[\mathbf{time}_{P'}\{P' \leftrightarrow V(x)\}\right]$$

□

6 Comparisons with Previous Definitions

One measure of the meaningfulness of our definition is provided by the fact that proofs of knowledge in our sense enjoy prover feasibility, a property that we have argued to be crucial in its own right. We may however ask whether other notions of proof of knowledge possess this property. The purpose of this section is to show that previous definitions do enjoy some weaker prover-feasibility property. For the purpose of this extended abstract, we exemplify this for the notion of Bellare and Goldreich. To avoid confusion we refer to their notion as a *liberal proof of knowledge*. Casting their definition in our syntax we get

Definition 3: [BG92]: An interactive Turing-machine V is a knowledge-verifier for a binary relation R with error ε if it satisfies the following

Completeness There exists an interactive Turing machine P (the "legitimate prover") such that for all $(x, y) \in R$, $\Pr[\mathbf{out}\{P(x,y) \leftrightarrow V(x)\} = YES] = 1$.

Validity with error ε There exists a probabilistic, expected polynomial-time Experimenting Machine K, so that for every $x \in \Sigma^*$ and every Interactive Turing-machine P', the result of $K(x)^{P'}$ is in $R(x) \cup \{\perp\}$ and

$$\Pr\left[\mathbf{result}\{K(x)^{P'}\} \in R(x)\right] \geq \Pr\left[\mathbf{out}\{P' \leftrightarrow V(x)\} = YES\right] - \varepsilon$$

If P is a "legitimate prover" (from the Completeness condition), we call the protocol (P, V) a *liberal proof of knowledge* for R (with error ε). An Experimenting machine K which satisfies the Validity condition is called a *liberal knowledge extractor* for the proof system (P, V).

REMARK. We argue that the main difference between this definition and ours is that in the above definition the knowledge extractor is not limited in the ways that it can communicate with an alleged prover P'. There are, however, a few other technical differences between these definitions.

1. Our definition is stated in terms of a protocol (P, V), while the above definition is stated in terms only of the verifier V (where the prover P is quantified in the Completeness condition). This difference is only a matter of presentation-style.

2. The Bellare-Goldreich definition allows the knowledge extractor K to run in *expected* polynomial time and requires that it succeeds with probability which is at least as large as the probability that the verifier is convinced. Instead, our definition requires that K be *strict* polynomial time, and allows its probability of success to be polynomially smaller. As long as the probability that the verifier is convinced is non-negligible, these two notions are equivalent, but otherwise they are incomparable (i.e., none of them implies the other). We chose the later form for our definition since we need it for the strong prover-feasibility property.[2] As we show below, for the weaker prover-feasibility property, the former choice suffices.

6.1 A Weaker Prover-Feasibility Property

Let (P, V) be a liberal proof of knowledge and let K be a liberal knowledge extractor for (P, V). It is clear that we cannot hope to prove that K maintains the feasibility of interacting with V, since as soon as K deviates in any way from the behavior of V, there exists a machine P' which runs for a very long time on exactly those conversations which happen with higher probability in the communication with K than in the communication with V. We therefore have to settle for a weaker notion of prover feasibility.

Intuitively, we show that a liberal knowledge extractor "does not quite need" the conversations which cause P' to run for a very long time. Namely, the extractor would still be able to extract the knowledge *if it were given access to a modified prover* which does not generate these long conversations. However, as we shall see, getting hold of such a modified prover may not be easy.

TRUNCATED PROVERS. To formalize the above intuition, we define a notion of a *truncated prover*. Informally, if P' is any Interactive Turing machine which may communicate with V, then the truncated version of P' (which we denote by $P'|$) is another Interactive Turing machine which behaves exactly like P', but has a time-out mechanism. Once the running-time of P' exceeds some threshold, the truncated version simply terminates its computation. Formally, we have

Definition 4: Let (P, V) be a liberal proof-of-knowledge system. Fix some $x \in \Sigma^*$, and some Interactive Turing-machine P' and denote $T' \stackrel{\text{def}}{=} E[\mathbf{time}_{P'}\{P' \leftrightarrow V(x)\} \mid \mathbf{out}\{V\}P' \leftrightarrow V(x) = YES]$ and $q' \stackrel{\text{def}}{=} \Pr[\mathbf{out}\{P' \leftrightarrow V(x)\} = YES]$.

The *truncated version of* P' (with respect to V, x), which we denote by $P'|$ it an Interactive Turing machine which in every execution behaves just like P' as long as the running-time of P' is less than $2T'$, and halts as soon as the running time exceeds this threshold.

[2] The reason for that is the well known problem with composition of expected polynomial-time algorithms.

The idea behind the following theorem is that if P' convinces V with a certain probability then $P'|$ must convinces V with similar probability, and therefore the success probability of the knowledge extractor K when talking to $P'|$ must be similar to the success probability while talking to P'. This argument is made formal in the following

Theorem 3: Let (P, V) be a liberal proof of knowledge, let K be a liberal knowledge extractor for (P, V). Then there exists some polynomial $Q(\cdot)$ so that for any $x \in \Sigma^*$ and any Interactive Turing-machine P', the truncated version of P' (with respect to V, x) satisfies

1. $E[\mathbf{etime}_{P'|}\{K(x)^{P'|}\}] \leq Q(|x|) \cdot 2T'$
2. $\Pr\left[\mathbf{result}\{K(x)^{P'|}\} \in R(x)\right] \geq \frac{q'}{2} - \varepsilon$

where $q' \stackrel{\text{def}}{=} \Pr[\mathbf{out}\{P' \leftrightarrow V(x)\} = YES]$ and
$T' \stackrel{\text{def}}{=} E[\mathbf{time}_{P'}\{P' \leftrightarrow V(x)\} \mid \mathbf{out}\{P' \leftrightarrow V(x)\} = YES]$.

Proof:
1. Since K is expected polynomial time then there exists a polynomial $Q(\cdot)$ such that the expected length of the e-transcript of $K(x)^{P'|}$ is bounded by $Q(|x|)$. The fact that $P'|$ never runs for more than $2T'$ steps in each execution completes the proof.

2. Since the expected running time of P' in an *accepting conversation* with V on x is T', then

$$\Pr\left[\mathbf{time}_{P'}\{P' \leftrightarrow V(x)\} > 2T' \mid \mathbf{out}\{V\}P' \leftrightarrow V(x) = YES\right] < \frac{1}{2}$$

which implies that

$$\Pr[\mathbf{out}\{P'| \leftrightarrow V(x)\} = YES] \geq \Pr[\mathbf{out}\{P' \leftrightarrow V(x)\} = YES] - \frac{q'}{2} \geq \frac{q'}{2}$$

and the theorem follows from the definition of K. □

6.2 Why Weaker

We note that the theorem above is weaker than Theorem 2 in several ways. Not only does it require that we modify the alleged prover P' (so as to omit the long conversations), but in fact this modification is highly non-uniform. Namely, the truncated P' depends not only on P' itself, but also on (the verifier and) the specific common input x. Even if one fixes an alleged prover P', there may be no single machine P'' that can play the role of $P'|$ for every input x.

One may try to overcome this non-uniformity of the truncated prover by experimenting with the original alleged prover P' in order to estimate the cutoff time T'. This, however, does not work. Assume, for example, that P' happens to runs in polynomial time on the accepting conversations, but only convinces V with a sub-polynomial probability. Then, even hitting a single accepting conversation of P' with V (let alone estimating the average running time of such conversations) takes time which is not polynomial in the running time of P'. (Of course, one may achieve uniformity, but at the expense of multiplying the running time of the truncated prover by a factor of $1/q'$.)

7 (Flawed) Alternative Definitions

Formulating our restrictions regarding the semantics of a proof of knowledge turned out to be a somewhat delicate matter. While we would like to restrict knowledge extractors in several ways to ensure that they are "semantically meaningful", we have endeavored to achieve the most general meaningful notion. Many ways to formulate our basic concerns, result in restricting this notion to the point of disallowing otherwise perfectly good

protocols. Without undergoing a complete analysis, let us demonstrate why two "seemingly simpler" notions fail.

EXAMPLE 1. As the first example, consider requiring that the EXTRACT function be applied only to transcripts (rather than to complete verifier-views. That is, after interacting with an alleged prover, the experimenting machine K discards the "verifier-coins" and passes to the EXTRACT function only the messages which were sent in the accepting conversations of this experiment. Although this may seem plausible, (and knowledge extractors of this form exist for most of the protocols in the literature), this restriction in fact rules out protocols which are otherwise perfectly good (and potentially useful).

To see that, take any proof of knowledge, and use it in a setting where the messages sent from the prover are *encrypted*. Namely, at the first round of the new proof, the verifier send a public-key to the prover, and the prover then uses this public key to encrypt all further communication with the verifier. Clearly, we would like to claim that if the original protocol was a proof of knowledge, then the new one is also a proof of knowledge. However, if we restrict the *extract* function to only see the messages, then in particular it will not have an access to the private key (since it was never sent). Since the *extract* function runs in polynomial-time, it will not be able to use the encrypted communication to extract the secret.

EXAMPLE 2. For a second example, consider demanding that the knowledge extractors generate a sequence of random *and independent* verifier views. That is, given that the coins of the alleged prover are fixed, all the verifier views should results from random and independent coins of the verifier. This notion too may seem plausible, and as in the case of our first example, knowledge extractors of this form exist for most of the protocols in the literature. However, here too we show that this requirement rules out protocols which should otherwise be considered valid proofs of knowledge.

Theorem 4: Under the quadratic-residuosity assumption, there exist a protocol which is a strong proof of knowledge, but does not satisfy the above more restrictive notion.

Proof (sketch): Consider modifying the Goldwasser-Micali-Rackoff protocol by changing the way the prover picks the initial random element $r \in z_n^*$. We add to the protocol a first round in which the verifier sends an $|n|$-bit random string u to the prover. The prover then picks another $|n|$-bit random string v and uses a *pseudo-random function* (in the sense of [GGM86]) to compute $r = f_v(u)$. The whole protocol is as follows:

	Prover $P(n, x, y)$	transcript	Verifier $V(n, x)$
1.		$\leftarrow u$	Pick a random $\|n\|$-bit string u.
2.	Pick a random $\|n\|$-bit string v. Compute $r = f_v(u)$, $s = xr^2 \bmod n$	$s \rightarrow$	
3.		$\longleftarrow \sigma$	Pick a random bit $\sigma \in \{0, 1\}$
4.	Compute $t = yr \bmod n$ if $\sigma = 1$ and $t = r$ otherwise	$t \rightarrow$	
5.			Check that $t^2 \equiv s \bmod n$ if $\sigma = 1$ and $xt^2 \equiv s \bmod n$ otherwise

It is obvious that this protocol constitute a strong proof of knowledge (with error $\frac{1}{2}$) for the quadratic residuosity, since the knowledge extractor can send the same (randomly selected) string u in all the conversations in its e-transcript and therefore extract the knowledge as before (See Section 4).

On the other hand, if we force K to send an independent random strings u in each conversation, then under the quadratic-residuosity assumption it only has negligible probability of extracting a square root of $x \bmod n$,

even when talking to the honest prover P. The reason is that if we replace f with a truly-random function, then generating many views of $P(n, x, y) \leftrightarrow V(n, x)$ is equivalent to running many independent executions of the Goldwasser-Micali-Rackoff protocol. Since this protocol is proved in [GMR85] to be zero-knowledge, it cannot help the knowledge extractor to compute a square root of x mod n. Finally, since the extractor is polynomial-time, it cannot tell the difference between a pseudo-random f and a truly random one and thus will fail to compute a square root of x mod n when f is pseudo-random. □

References

[BG92] O. Goldreich and M. Bellare, "On Defining Proofs of Knowledge". In *Advances in Cryptology: CRYPTO '92,*, Lecture Notes in Computer Science, volume 740, Springer-Verlag, 1992. Pages 390-420

[FFS87] U. Feige and A. Fiat and A. Shamir. "Zero-knowledge proofs of identity". In *Journal of Cryptology*, 1(2), 1988. Pages 77-94

[FMR84] M.J. Fischer, S. Micali and C. Rackoff, "A Secure Protocol for Oblivious Transfer (Extended Abstract)" In *Journal of Cryptology*, 9(3), 1996. Pages 191-195 (first presented in Eurocrypt '84).

[FS90] U. Feige and A. Shamir. "Witness indistinguishable and witness hiding protocols". In *Proc. of the 22nd Annual ACM Symposium on Theory of Computing*, 1990. Pages 416-426

[GGM86] O. Goldreich, S. Goldwasser and S. Micali. How to Construct Random Functions. In *Journal of the ACM*, Vol. 33, no. 4, 1986, pp. 792-807

[GMR85] S. Goldwasser, S. Micali and C. Rackoff. "The knowledge complexity of interactive proof systems". In *SIAM Journal on Computing*, 18(1), 1989. Pages 186-208 (Preliminary version appeared in Proc. of the 17th Annual ACM Symposium on Theory of Computing, 1985. Pages 291-304)

[TW87] M. Tompa and H. Woll. "Random self-reducibility and zero knowledge interactive proofs of possession of information". In Technical report CS92-244, Univ. of California at San-Diego, 1992. (Preliminary version appeared in 28th Annual Symposium on Foundations of Computer Science, IEEE, 1987. Pages 472-482)