More on Proofs of Knowledge

Shai Halevi* Silvio Micali†

May 4, 1998

Abstract

The notion of *proofs of knowledge* is central to cryptographic protocols, and many definitions for it have been proposed. In this work we explore a different facet of this notion, not addressed by prior definitions. Specifically, prior definitions concentrate on capturing the properties of the verifier, and do not pay much attention to the properties of the prover.

Our new definition is strictly stronger than previous ones, and captures new and desirable properties. In particular, it guarantees *prover feasibility*, that is, it guarantees that the time spent by the prover in a proof of knowledge is comparable to that it spends in an "extraction" of this knowledge. Our definition also enables one to consider meaningfully the case of a single, *specific* prover.

Keywords: Proofs of knowledge, prover feasibility.

1 Introduction

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.

Proofs of knowledge have since played a crucial role in cryptographic protocols, and ever since their intuitive introduction [6] [3], there have been several attempts to provide them with an adequate formalization. The first such attempts [4, 2, 8] did not address some possible scenarios (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). These weaknesses were observed and corrected in a later work of Bellare and Goldreich [1].

Although this last definition captures many important aspects of proofs of knowledge, we argue below that there are still other aspects which are not addressed by it. In this paper we exhibit one such aspect and propose to strengthen previous definitions so as to capture it.

THE COMMON CORE OF PRIOR DEFINITIONS. For the sake of simplicity, in the intuitive discussion below we denote the common input to the prover and verifier by x and its corresponding 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

*IBM Watson, P.O. Box 704, Yorktown Height, NY 10598, USA, shaih@watson.ibm.com

†MIT - Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139, USA

with P a few times, outputs a secret y. (One instructing way to think of this knowledge extractor is as the "sub-conscience" of the prover. Thus, if the prover convinces the verifier that it knows y, then it can also reconstruct y by itself.)

On the other hand, often proofs of knowledge are 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 such contradiction occurs is that all prior notions assume that the prover is *resettable*; namely, it is possible to restore the provers state at the beginning of the computation (including the contents of its random tape, should the prover be a probabilistic Turing machine).

1.1 Our Notion of Proofs Of Knowledge

Assume that a prover P interacts with a Verifier V on a common input x, and succeeds in convincing 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 the conversation with P. Notice now that this view is actually a *probability distribution* that it determined by both P and V (and their probabilistic choices). Therefore, it must be *this* distribution that demonstrates P's knowledge of y.

However, prior definitions allow that a knowledge extractor K 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.

THE SAME-DISTRIBUTION CONSTRAINT. Our notion closes the above "semantic gap" by demanding that each interaction of the extractor K with the prover P has exactly the same distribution of the prover-verifier interactions. We call a knowledge extractor that obeys this 'same-distribution' constraint a *conservative knowledge-extractor*, and the resulting notion a *conservative proof of knowledge*.

We stress that our definition, while insisting that each single K-P interaction is distributed just like a random P-V interaction, *allows these interactions to be dependent.*

It is easy to verify that all the known proof-of-knowledge protocols in the literature indeed have conservative knowledge extractors. (But it is still important to find a better way of capturing what these protocols actually accomplish.)

PROVER PROPERTIES AND SPECIFIC PROVERS. The 'same distribution' constraint essentially tells us that the behavior of the prover when interacting with the extractor is the same as when interacting with the verifier. Hence, it enables us to make assertions regarding the properties of the prover in a proof-of-knowledge. In particular, it enables us to ask meaningfully questions of the form "does a particular prover-program P demonstrates a knowledge of some secret?"

We note that this is in sharp contrast to previous definitions, where provers are quantified under the 'for all' quantifier and referring to a specific prover-program is quite meaningless. To demonstrate this point, let (P, V) be any proof of knowledge protocol, and consider the following contrived modification of the prover-program.

Program P': On common input x and secret input y' (which may or may not be a 'valid secret' with respect to x), behave just like P does, except when the verifier's query consists of the all-zero string. In this case, perform an exponential-time exhaustive search for a valid secret y, and sends it to the verifier.

It is clear that to extract y from such a contrived prover P' one may just send it the all-zero string, but it is also clear that such extraction of y has little relation to the 'knowledge' of y that P' demonstrated in an interaction with the verifier (when the all-zero string is not sent). Nonetheless, previous definitions of proofs-of-knowledge were formulated in a way which did not enabled us to make this distinction.

We note that this is no accident. Indeed, previous definitions were explicitly formulated to capture only the properties of the verifier. In particular, the definition in [1] goes as far as defining a proof of knowledge only in terms of the verifier. In this paper, we argue that by adding the 'same distribution' constraint, we enforce desirable properties of the prover as well. An important example is the prover-feasibility property, which is discussed below.

Prover Feasibility. As the notion of running-time is a central one for Turing-machines, we argue that it is important that the amount of time it takes P to convince V that it knows y be comparable to the time needed to extract y from P. Indeed, in many proofs of knowledge, the secret y can typically 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 do not address the time that P may invest when interacting with K in order to compute y. 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 stated, this property means the following:

While extracting y, the total running-time of both the knowledge-extractor and the prover is comparable to the running-time of the prover when interacting with the verifier.

We formally define prover feasibility in Section 4, and prove that it is automatically satisfied by our stronger notion of proof of knowledge. In Section 5 we show that prior notions only satisfy a *weaker* prover-feasibility property. Moreover, they do not handle the feasibility of a single prover: they satisfy this weaker prover-feasibility property only when "quantifying over all provers".

De-coupling the knowledge extractors. So far, we have argued that, for (P, V) to be a proof of knowledge, the secret y should be efficiently computable not just by interacting with (and resetting) P in any way we want, but in a way that produces a few, random P-V interactions. We now further specify that y should be *manifest in the verifier's views* corresponding to these interactions.

Indeed, when the prescribed Verifier V interacts with P and becomes convinced that she knows y, he reaches this conclusion based solely on his own view, that is, the coins he tossed during the interaction and the messages received from P. We thus demand that y be computable from these views *alone*.

Notice now that, if some machine M exchanges messages with P just as V does and then computes y, it could do so thanks to some information encoded in its own internal state, information that may not be derivable from just V's views.

Accordingly, we require that the process of computing y have two distinct components. The first is an efficient *sampler* that interacts with (and resets) P so as to obtain views (coins and messages) that are distributed exactly as the views of V interacting with P. The second is an efficient *extractor* that computes the desired y from just such views.

Thus, for the purposes of computing y, the sampler does not furnish the extractor with information about its own internal state. This ensures that y is manifest in the views themselves, and independent of the specific nature of samplers and extractors (though for concreteness we let them be polynomial-time Turing machines).[1]

ORGANIZATION OF THE PAPER. 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 *conservative proof-of-knowledge.* In Section 4 we define the notion of *prover feasibility* and prove that our definition enjoys that property. In Section 5 we show that prior definitions possess a weaker notion of prover feasibility.

2 Preliminaries

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.

INTERACTIVE TURING MACHINES. We rely on the intuitive notion of an interactive Turing-machine (ITM) as a machine which can send and receive messages during its computation. The reader is referred to [6] for a more formal definition. A pair of interactive Turing machine (P, V) defines a *protocol* in the usual way. For convenience, given our application, we refer to the first machine in this pair as the "Prover" and to the second machine as the "Verifier".

TRANSCRIPTS AND VIEWS. Let (P, V) be a protocol and let (a, b) be the input to this protocol (so that a is a private input to P, and b is a private input to V). Consider an execution of (P, V) on input (a, b); then, the sequence of messages exchanged between P and V in this execution is called *the transcript* of the execution. A verifier-view for this execution consists of the transcript of the execution and V's coin tosses in it. We denote by $\mathbf{view}_V\{P(a) \leftrightarrow V(b)\}$ the distribution of the verifier-views in a random execution of protocol (P, V) on input (a, b).

RUNNING-TIME. With each execution of (P, V) on input (a, b), we consider the running-time of the Prover P during this execution. If D is any distributions over executions, then we denote by $\mathbf{time}_P\{D\}$ the corresponding distribution of the running time of P.

ACCEPTANCE. We say that an execution of (P, V) on (a, b) in *accepting*, if at the end of the execution the Verifier V enters a designated accept state. The event of accepting execution is denoted $\mathbf{accept}\{P(a) \leftrightarrow V(b)\}$.

ITM-ORACLE MACHINES. Here too we rely on the intuitive notion of an ITM having as an oracle another ITM. We denote an ITM-oracle machine K with access to an interactive machine P by K^P. K^P has the ability to interact with P several times, each time resetting it to its initial configuration (which includes the original random tape).

[1]Notice that such model independence might not arise if y were computed by a single efficient component, for instance, by a single efficient Turing machine M interacting with P. In this case, in fact, M could use its own internal state — e.g., the content of its working tapes — for computing y, and internal states are not easily "exportable" across different models of computation.

BINARY RELATIONS. We prove knowledge about polynomially-bounded binary relations. Let $R \subset \Sigma^* \times \Sigma^*$ be a binary relation. We say that R is *polynomially-bounded* if there is a polynomial Q such that, for all $(x, y) \in R$, $|y| \leq Q(|x|)$. For every string x we denote the set of strings which are R-related to x by $R(x)$. Namely, $R(x) \overset{\text{def}}{=} \{y : (x, y) \in R\}$.

3 Conservative Proofs of Knowledge

Our notion of proof of knowledge is based on the syntactic concept of a valid experiment.

VALID DISTRIBUTIONS. Let Q be any polynomial, (P, V) be a protocol and x an input. (Below, we consider x to be an input to the Verifier only because, for future purposes, a Prover's input can always be "wired in" P's description.) Consider running protocol (P, V) on input x for $Q(|x|)$ times, each time giving the Prover the same random tape r_P, and consider the sequence of Verifier-views thus obtained. We say that a distribution $\mathcal{D}$ over such sequence is *valid (P,V,x,Q)-distribution* if

(*a*) r_P has been randomly selected, and

(*b*) each entry in the sequence is distributed as the Verifier's view in an execution of (P, V), with input x, where the Prover's tape is r_P and the Verifier's tape is uniformly selected.

Of course, one way of generating $\mathcal{D}$ consists of running protocol (P, V) on input x for $Q(|x|)$ times, each time using r_P as the Prover's tape and an independently and uniformly selected tape for the Verifier. Notice, however, that this is not the only way. Indeed, in a sample according to $\mathcal{D}$, *although each Verifier-view is uniformly selected, the views may be dependent.*

OUR DEFINITION. 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. Following [1], 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 *Conservative Proof of Knowledge* with error ε for a binary relation R, consists of a protocol (P, V) with the following properties:

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

Validity with error ε. There exists a polynomial-time ITM-oracle machine K, a polynomial-time algorithm EXTRACT, a constant $c > 0$, and a polynomial Q such that, for every $x \in \Sigma^*$ such that $R(x) \neq \emptyset$ and every Interactive Turing machine P',

1 $K^{P'}(x)$ is a valid (P', V, x, Q)-distribution, and

2 $\text{EXTRACT}(K^{P'}(x)) \in R(x) \cup \{\perp\}$, and

3 $\Pr\left[\text{EXTRACT}(K^{P'}(x)) \in R(x)\right] \geq (p - \varepsilon)^c$

where $p \overset{\text{def}}{=} \Pr\left[\mathbf{accept}\{P' \leftrightarrow V(x)\}\right]$.

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

REMARKS:

1. 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.

2. Following [1], we too define the validity condition only for strings x such that $R(x) \neq \emptyset$. The reader is referred to [1] for a discussion of this choice.

3.1 The case of a specific prover.

One advantage of the above definition is that it enables us to talk about the extent in which the verifier's view of a conversation with a *specific prover* demonstrates knowledge of a secret.

Definition 2: Let V be a verifier in a conservative proof of knowledge for a relation R, let P' be some specific prover, and let $\mathbf{view}_V\{P' \leftrightarrow V(x)\}$ be the distribution of conversations between V (with input x) and the specific prover P'. We say that this distribution *demonstrates knowledge* of some $y \in R(x)$ with probability p, if the accepting views have probability weight of p in this distribution.

We stress that Definition 1 implies that as long as $p \geq \varepsilon$, one can indeed extract an element $y \in R(x)$ from a few views which are sampled according to the distribution $\mathbf{view}_V\{P' \leftrightarrow V(x)\}$.

By contrast, the reader may notice that (like for all prior definitions) adapting the definition of proof of knowledge due to Bellare and Goldreich to a single prover P' is not meaningful (as discussed in our introduction).

4 Conservative Proofs of Knowledge Are Prover-Feasible

In this section we show that conservative 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 3: Let V be an interactive Turing machines and let K be an ITM-oracle 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

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

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

Theorem 1: Let (P, V) be a conservative proof of knowledge system for the binary relation R, and let (K, 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'. Since $K^{P'}(x)$ generates a valid (P', V, x, Q)-distribution, then each instance of $K^{P'}(x)$ consists of exactly $Q(|x|)$ Verifier-views. Moreover, for $i = 1, 2, \cdots, Q(|x|)$, the distribution of the i'th Verifier-view in $K^P(x)$ is identical to the distribution of the verifier-view in a random execution of (P', V) on x. Therefore,

$$E\left[\mathbf{time}_{P'}\{K^{P'}(x)\}\right] = Q(|x|) \cdot \mathbf{time}_{P'}\{P' \leftrightarrow V(x)\}$$

□

5 A Weak Version Of Prover Feasibility

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 (though it may be also enjoyed by other prior notions). To avoid confusion we refer to their notion as a *liberal proof of knowledge*.

5.1 The Bellare-Goldreich Definition

Casting their definition in our syntax we get

Definition 4: [1]: 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 $(x, y) \in R$, $\Pr[\mathbf{accept}\{P(x, y) \leftrightarrow V(x)\}] = 1$.

Validity with error ε There exists a probabilistic, expected polynomial-time ITM-oracle machine K, so that for every $x \in \Sigma^*$ such that $R(x) \neq \emptyset$ and every Interactive Turing-machine P', $K^{P'}(x) \in R(x) \cup \{\perp\}$ and

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

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 ITM-oracle 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.

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

5.2 A Weaker 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 5: Let (P, V) be a liberal proof-of-knowledge system. Fix some $x \in \Sigma^*$, and some Interactive Turing-machine P' and denote $T' \overset{\text{def}}{=} E[\textbf{time}_{P'}\{P' \leftrightarrow V(x)\} \mid \textbf{accept}\{V\}P' \leftrightarrow V(x)]$ and $q' \overset{\text{def}}{=} \Pr[\textbf{accept}\{P' \leftrightarrow V(x)\}]$.

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.

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 2: 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. $E[\textbf{time}_{P'|}\{\text{executions of } K(x)^{P'|}\}] \leq Q(|x|) \cdot 2T'$

2. $\Pr\left[K(x)^{P'|} \in R(x)\right] \geq \frac{q'}{2} - \varepsilon$

where $q' \overset{\text{def}}{=} \Pr[\textbf{accept}\{P' \leftrightarrow V(x)\}]$ and
$T' \overset{\text{def}}{=} E[\textbf{time}_{P'}\{P' \leftrightarrow V(x)\} \mid \textbf{accept}\{P' \leftrightarrow V(x)\}]$.

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

$$\Pr[\textbf{time}_{P'}\{P' \leftrightarrow V(x)\} > 2T' \mid \textbf{accept}\{P' \leftrightarrow V(x)\}] < \frac{1}{2}$$

which implies that

$$\Pr[\mathsf{accept}\{P'| \leftrightarrow V(x)\}] \geq \Pr[\mathsf{accept}\{P' \leftrightarrow V(x)\}] - \frac{q'}{2} \geq \frac{q'}{2}$$

and the theorem follows from the definition of K. □

5.3 Why Weaker

We note that the theorem above is weaker than Theorem 1 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 $1/q'$.)

6 In Sum

In sum, we provide a new definition of a proof of knowledge with a tighter semantics and achieving some desirable properties (such as prover feasibility). Our definition does not rule out prior proof-of-knowledge protocols, but provides a more satisfying explanation of why they "work". It wants, however, to rule out the possibility that one may design a formally correct but "semantically wrong" proof of knowledge in the future.

References

[1] 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

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

[3] 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).

[4] 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

[5] 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

[6] 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)

[7] O. Goldreich, S. Micali and A. Wigderson, "Proofs that Yield Nothing but their Validity or All Languages in NP Have Zero-Knowledge Proof Systems", *Journal of the ACM*, vol. 38, no. 1, 1991, pp. 691-729.

[8] 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)