Proving Computational Ability

Mihir Bellare* Oded Goldreich†

August 1992

(Unpublished draft)

Abstract

We investigate extending the notion of a proof of knowledge to a proof of the ability to perform some computational task. We provide some definitions and protocols for this purpose.

* Department of Computer Science & Engineering, Mail Code 0114, University of California at San Diego, 9500 Gilman Drive, La Jolla, CA 92093. E-mail: mihir@cs.ucsd.edu

† Department of Applied Mathematics and Computer Science, Weizmann Institute of Science, Rehovot, Israel. e-mail: oded@wisdom.weizmann.ac.il. Research was partially supported by grant No. 89-00312 from the US-Israel Binational Science Foundation (BSF), Jerusalem, Israel.

1 Introduction

We extend the idea of proving "knowledge" of a string to encompass a notion of "proving the ability to perform some task." Specifically, we wish to formalize what it means to "prove the ability to compute a function f on some instance distribution D."

Motivation. Such a notion might have many uses, and two of them are described here. Suppose `Alice` possess a trapdoor, $t(x)$, to a (publically known) trapdoor permutation f_x and wishes to identify herself to `Bob`, by demonstrating ability to invert f_x. The proof of ability should be zero-knowledge so to prevent `Bob` from latter impersonating `Alice`. Admittingly, in this case `Alice` can establish her identity by directly proving, in a zero-knowledge manner, her knowledge of the trapdoor $t(x)$ (which corresponds to the index x of f_x). Still it may be cheaper to prove ability to invert f_x (e.g., by using a trivial protocol in which the prover inverts f_x on instances chosen by the verifier). This is particularly valid in case `Alice` posseses special purpose hardware, in which the trapdoor is hard-wired, making it very easy for her to invert the function on inputs of her choice. A second application is for a party to prove possesion of vast computing power by conducting very difficult tasks (e.g., inverting one-way functions).

Related work. This is an extension of our previous work on proofs of knowledge [1] in which we try to generalize those ideas to the setting of proving computational ability.

Proofs of knowledge are first mentioned in [5] and have been seeing definitional refinements [3, 6, 2] culminating in the notions of [1, 4]. We assume the reader is slightly familiar with the notion.

Proofs of computational ability were first discussed by Yung [7]. We adhere to the same basic and natural idea (namely, that computational ability of a prover is certified if some extractor can use the prover as a black box to solve the problem itself) but our approach is more general. For example whereas an assumption on the problem hardness is made in [7] it is not made here; we consider notions of distribution-free and distribution-dependent ability; following [1] we define an analogue of "knowledge error"; and following [1] we avoid some weaknesses inherited from earlier definitions of proofs of knowledge.

2 Definitions

For greater generality, we will consider relations rather than functions. By a *family of relations* we mean a sequence $\{R_x\}_{x\in\{0,1\}^*}$, where $R_x \subseteq \{0,1\}^{|x|} \times \{0,1\}^*$ for each x. For simplicity we restrict our attention to polynomially bounded families; that is, we assume there is a polynomial p such that $(z, y) \in R_x$ implies $|z| = |x|$ and $|y| \leq p(|x|)$. Following the notation used in [1], we denote $R_x(z) \stackrel{\text{def}}{=} \{\, y : (z, y) \in R_x \,\}$ and $L_{R_x} \stackrel{\text{def}}{=} \{\, z : \exists y \text{ such that } (z, y) \in R_x \,\}$. Prover and verifier will interact on common input x, with the goal of the interaction being for the prover to "convince" the verifier that he has the "ability to solve R_x."

We need to address the meaning of both of the phrases in quotes above. We will first define what it means for a machine to "solve a relation" (or a family of relations), and only next will we define what is a "proof of ability" to do so.

The standard meaning of efficiently solving a relation, The standard meaning of solving a relation, $S \subseteq \{0,1\}^* \times \{0,1\}^*$, is the existence of an efficient algorithm that, on input z, outputs $y \in S(z)$, called a *solution* to z, if such exists. This is a notion of worst case. Instead, we adopt a notion of average case by which we consider a probability distribution on the inputs and require that the algorithm is efficient on the average (with respect to the input distribution). An even more liberal notion is derived by allowing the solver to ask for alternative inputs, which are generated according to the same distribution (and independently of previous inputs), until it can present a solution to any of the inputs.

Notation: Let $S \subseteq \{0,1\}^* \times \{0,1\}^*$. Then $\mathrm{dom}(S) \stackrel{\mathrm{def}}{=} \{\, z \in \{0,1\}^* : S(z) \neq \emptyset \,\}$ is the *domain* of S.

Definition 2.1 (Solving relations) *Let $S \subseteq \{0,1\}^* \times \{0,1\}^*$ be a relation, and D be a distribution on $\mathrm{dom}(S)$. Suppose $t \in \mathsf{N}$ and let $M(\cdot)$ be a machine.*

- *We say that* machine $M(\cdot)$ solves S under D in expected t steps *if, on input $(z_1, z_2, ..., z_t)$, with each z_i drawn independently according to D, machine M halts within expected t steps and outputs a pair (z_i, y) so that $y \in S(z_i)$. (The expectation here is over the random choices of M as well as the t-product of the distribution D.)*
- *We say that* machine $M(\cdot)$ strongly solves S under D in expected t steps *if, on input z, drawn according to D, machine M halts within expected t steps with output $y \in S(z)$. (The expectation here is over the random choices of M as well as the distribution D.)*

Conventions: If a machine has several inputs, we may fix some of them to obtain a machine on the remaining inputs. Likewise, for an oracle machine, we may fix the oracle and consider the resulting machine. Specifically, suppose that the oracle machine $M(\cdot,\cdot,\cdot)$ has three inputs, then $M^A(x,y,\cdot)$ denotes the machine with one input whose output on input z is $M^A(x,y,z)$.

Let $\mathcal{R} = \{R_x\}_{x\in\{0,1\}^*}$ be a family of relations. We say that $\mathcal{D} = \{D_x\}_{x\in\{0,1\}^*}$ is an input distribution for $\mathcal{R}$ if for every x, it holds that D_x is a distribution on $\mathrm{dom}(R_x)$. We are now ready to define proofs of ability to solve (repectively, ability to strongly solve) a family of relations under a family of distributions.

Definition 2.2 (Proof of ability) *Let $\mathcal{R} = \{R_x\}_{x\in\{0,1\}^*}$ be a family of relations, and $\mathcal{D} \stackrel{\mathrm{def}}{=} \{D_x\}_{x\in\{0,1\}^*}$ be an input distribution for $\mathcal{R}$. Let $\kappa\colon \{0,1\}^* \to [0,1]$. We say that an interactive function, V, is a* verifier of the ability to solve (resp., strongly solve), *$\mathcal{R}$* under *$\mathcal{D}$ with* error *κ if the following two conditions hold.*

- non-triviality: *There exists an interactive function P^* so that for all x, all possible interactions of V with P^* on common input x are accepting (i.e.* $\Pr[\mathrm{tr}_{P^*,V^{D_x}}(x)\in \mathtt{ACC}_V(x)] = 1$*).*
- validity: *There exists a constant $c > 0$ and a probabilistic oracle machine $K(\cdot,\cdot,\cdot)$ such that for every interactive function P, every $x \in \{0,1\}^*$ and every $\gamma \in \mathtt{ACC}_V(x)$, machine $K^{P_x}(x,\gamma,\cdot)$ satisfies the following condition:*

  if $p(x) \stackrel{\mathrm{def}}{=} \Pr[\mathrm{tr}_{P,V^{D_x}}(x) \in \mathtt{ACC}_V(x)] > \kappa(x)$ then machine $K^{P_x}(x,\gamma,\cdot)$ solves (resp., strongly solves) *R_x under D_x in an expected number of steps bounded by*

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

The oracle machine K is called an ability extractor (resp., strong ability extractor) *under $\mathcal{D}$.*

Hence an *ability extractor* is given a sequence of instances, each independently selected according to D_x, and is supposed to output a solution to one of these instances within the specified (expected) time bound. A *strong ability extractor* is given a single instance, selected according to D_x, and is supposed to output a solution to this instances within the specified (expected) time bound.

Proofs of knowledge (as per [1, Definition 3.1]) are a special case of proofs of ability. To justify this claim, given a binary relation R we define the family of relations $\mathcal{R} = \{R_x\}$ so that $R_x = \{\,(x,y) : (x,y) \in R\,\}$. Clearly, $\mathrm{dom}(R_x)$ is the singleton $\{x\}$ if $R(x) \neq \emptyset$ and $\emptyset$ otherwise. Let

D_x be the distribution on dom(R_x) which, in the former case, assigns the entire probability mass to x (and is undefined in the latter case). Clearly $\mathcal{D} = \{D_x\}$ is an input distribution for $\mathcal{R}$. It is easy to see that if V is a verifier of the ability to solve $\mathcal{R}$ under $\mathcal{D}$ (with error κ) then V is also a knowledge verifer for R (with knowledge error κ).

Definition 2.2 refers to a specific input distribution. Clearly, both the ability-verifier and the ability-extractor may depend on this distribution, and this dependency seems inevitable. However, the dependency on the input distribution can be "uniform" in the sense that both verifier and extractor can be fixed machines with access to a random source which generates the input distribution. We call such a proof of ability *distribution-free.*

Convention: Let $\mathcal{D}$ be a family of distributions for some $\mathcal{R}$, and let M be an (interactive and/or oracle) probabilistic machine. A *$\mathcal{D}$-source augmentation* of machine M is a machine that, on input x, in addition to the standard behaviour of M can obtain elements draw independently from distribution D_x (at the cost of reading them).

Definition 2.3 (Distribution-free proof of ability) *Let $\mathcal{R} = \{R_x\}_{x\in\{0,1\}^*}$ be a family of relations, and let κ: $\{0,1\}^* \to [0,1]$.*

- *We say that an interactive machine, V, is a* distribution-free verifier of the ability to solve $\mathcal{R}$ *with* error κ *if for every input distribution, denoted $\mathcal{D}$, for $\mathcal{R}$, the $\mathcal{D}$-source augmentation of machine V constitutes a verifier of the ability to solve $\mathcal{R}$ under $\mathcal{D}$ with error κ.*
- *We say that a distribution-free verifier of the ability to solve $\mathcal{R}$ (with error κ) has a* distribution-free ability extractor *if there exists an oracle machine, K, so that the $\mathcal{D}$-source augmentation of machine K constitutes a ability extractor under $\mathcal{D}$.*

A definition of a *distribution-free strong ability extractor* is derived analogously.

3 Examples

To demonstrate the above definitions we consider two natural examples. Both examples refer to a familty of trapdoor one-way permutations, $\{f_x\}_{x\in\{0,1\}^*}$. The string x is called the index of the permutation $f_x : \{0,1\}^{|x|} \mapsto \{0,1\}^{|x|}$. There exists an efficient algorithm that, on input index x and argument y, returns the value $f_x(y)$. There is an efficient algorithm that, on input index x and value v, returns the inverse of f_x on v (i.e., $f_x^{-1}(v)$).

Example 1: Consider a verifier that, on common input x, sends the prover a single uniformly selected string $v \in \{0,1\}^{|x|}$, and accepts if and only if the prover answers with the inverse of v under f_x (i.e., with y satisfying $f_x(y) = v$). We show (below) that the above verifier is an ability-verifier for inverting f_x under the uniform distribution.

Example 2: Consider a verifier that, on common input $x \in \{0,1\}^n$ ($n \in \mathsf{N}$), sends the prover $2n$ uniformly and indepedently selected strings, $v_1, ..., v_{2n} \in \{0,1\}^n$, and accepts if and only if the prover answers with the inverse of each of these v_i's under f_x (i.e., with $y_1, ..., y_{2n}$ satisfying $f_x(y_i) = v_i$, for every i). We show (below) that the above verifier is a *strong* ability-verifier for inverting f_x on at least one out of $2|x|$ of uniformly selected instances.

Proposition 3.1 *The program described in Example 1 is an ability-verifier (with error zero) for solving $\mathcal{R} = \{R_x\}$ under $\mathcal{D} = \{D_x\}$, where*

- *$R_x = \{(v,y) : v = f_x(y)\}$;*

- *D_x is uniform over the set of all strings of length $|x|$.*

Furthermore, if the verifier in Example 1, selects v according to an arbitrary distribution D_x, then the system described constitutes a distribution-free proof of ability.

proof sketch: We present here only the case of uniform distribution. Consider an arbitrary, fixed prover. Let p_x denote the probability that the verifier is convinced on common input x. Here the probability space is over all choices of both the verifier and prover. Assume, without loss of generality, that $p_x > 2^{-|x|}$, otherwise the extractor satisfies the requirement by merely exhaustive search. Also, we may assume that the ability-extractor "knows" p_x since it may estimate p_x in expected time $\text{poly}(x)/p_x$ by repeated experiments. Let $q_x(v)$ denote the probability that the verifier is convinced conditioned on the event that it chose and sent v to the prover. Here the probability distribution is merely over the prover's random coins (in case it is at all probabilistic). Let $V_x(i)$ be the set of v's for which $q_x(v)$ is greater than 2^{-i} and smaller/equal to 2^{-i+1}. Clearly, there exists an $i \leq |x|$ such that

$$\frac{|V_x(i)|}{2^{|x|}} > \frac{p_x \cdot 2^i}{n} \tag{1}$$

We are now ready to present the ability-extractor. Formally speaking, the extractor gets as input an index, x, and a sequence of independently and uniformly selected $|x|$-bit long strings, and its task is to invert f_x on one of them. However, to simplify the exposition, we prefer to think of these strings as being chosen by the extractor. Hence, on input x, the extractor executes $m \stackrel{\text{def}}{=} \lceil \log_2(1/p_x) \rceil$ copies of the following procedure, each with a different value of $i \in \{1, ..., m\}$. The i^{th} copy consists of uniformly and independently selecting $M \stackrel{\text{def}}{=} \text{poly}(n)/(p_x \cdot 2^i)$ values, $v_1, ..., v_M$, and executing the following sub-procedure on each of them. The sub-procedure with value v_j invokes the prover's program (as oracle), on input x and message v_j, for $\text{poly}(n) \cdot 2^i$ times, each time checking whether the prover's answer is the inverse of v_j under f_x. Once a positive answer is obtained, the extractor halts with the corresponding value-inverse pair.

The extractor's expected running-time is bounded above by

$$\sum_{i=1}^{m} \frac{\text{poly}(n)}{p_x \cdot 2^i} \cdot (\text{poly}(n) \cdot 2^i) = \frac{\text{poly}(n)}{p_x}$$

To evaluate the performace of the above extractor, consider the i^{th} copy, where i satisfies Equation (1). With overwhelmingly high probability (i.e., greater than $1 - 2^{-n}$), one of the v_j's chosen in this copy satisfies $q_x \geq 2^{-i}$. Hence, with overwhelmingly high probability, the extractor inverts f_x on this v_j. The exponentially small error probabilities can be eliminated by running an exhaustive search algorithm (for inverting f_x) in parallel to the entire algorithm described above. The proposition follows. $\square$

Proposition 3.2 *The program described in Example 2 is a strong ability-verifier for solving $\mathcal{R} = \{R_x\}$ under $\mathcal{D} = \{D_x\}$, where*

- *$R_x = \{(v_1, ..., v_{2|x|}, y) : \exists i \text{ s.t. } v_i = f_x(y)\}$;*
- *D_x is uniform over the set of strings of length $2|x|^2$.*

proof sketch: As in the proof of Proposition 3.1, we consider an arbitrary fixed prover and let p_x denote the probability that the verifier is convinced on common input x. As before, we may assume that $p_x > 2^{-|x|}$ and that the ability-extractor has a good estimate of p_x. Let $n \stackrel{\text{def}}{=} |x|$, and consider an $2n$-dimentional table in which the dimensions correspond to the $2n$ values chosen by the verifier. The $(v_1, ..., v_{2n})$-entry in the table equals the probability that the prover convinces the verifier (i.e., successfuly inverts f_x on v_1 through v_{2n}) conditioned on the event that the verifier sent message $(v_1, ..., v_{2n})$ to the prover. The probability here is merely on the prover's random choices. As in the proof of Proposition 3.1, we consider a partition of these probabilities to clusters of similar magnitude. It follows that there exists an $m < 2n$ such that at least a $q_x \stackrel{\rm def}{=} p_x \cdot 2^m/2n$ fraction of the entries have value greater than 2^{-m}. We call these entries *admisible*. It follows that there exists a dimention i so that at least a $\sqrt[2n]{q_x/2} > \frac{1}{2}$ of the rows in the $i^{\rm th}$ dimention contain at least $q_x/2n$ admisible entries. We call such a (m, i) pair *good*.

We are now ready to present the *strong* ability-extractor. The extractor gets as input an index, x, and a uniformly chosen $2|x|^2$-long string $\overline{v} = (v_1, ..., v_{2n})$, where $v_j \in \{0,1\}^n$ and $n = |x|$. The extractor is suppose to find a solution to $\overline{v}$, and this amounts to inverting f_x on one of the v_j's. To this end the extractor executes $8n^3$ copies of the following procedure, each with a different triples (m, i, j), where $1 \leq m, i, j \leq 2n$. The $(m, i, j)^{\rm th}$ copy of the procedure tries to invert f_x on v_j, using the parameters i and m. Specifically, the $(m, i, j)^{\rm th}$ copy consists of repeatedly invoking the sub-procedure $A_{m,i}$ on input v_j, for at most $\lfloor {\rm poly}(n)/(p_x \cdot 2^m) \rfloor$ times.

On input v, the sub-procedure $A_{m,i}$ proceeds as follows.

1. Selects uniformly $2n$ strings of length n each. These strings are denoted $u_1, ..., u_{2n}$;
2. Invokes the (oracle to the) prover ${\rm poly}(n) \cdot 2^m$ times, each time with input x and verifier's message $(u_1, ..., u_{i-1}, v, u_{i+1}, ..., u_{2n})$. The message consist of the sequence selected at step (1), except that u_i is replaced by v.
3. If in one of these invocations, the prover answers with a $2n$-tuple $(y_1, ..., y_{2n})$ such that $f_x(y_i) = v$ then the extractor halts with output (v, y_i).

Clearly, the expected running-time of the above extractor can be bounded by ${\rm poly}(x)/p_x$. To evaluate the performance of the above extractor, consider a good pair (m, i). By definition of a good pair, it follows that at least one half of the rows in the $i^{\rm th}$ direction contain at least $\rho_{m,i} \stackrel{\rm def}{=} p_x \cdot 2^m/(2n)^2$ entries on which the prover convinces the verifier with probability at least 2^{-m}. Let us denote the set of n-bit strings corresponding to these rows by S_x. It follows that for every $v \in S_x$, the sub-procedure $A_{m,i}$ inverts f_x on v with probability at least $\rho_{i,m} - 2^{-n}$. Hence, when invoking $A_{m,i}$ on v for ${\rm poly}(n)/\rho_{m,i}$ times, with overwhelming probability (i.e., greater than $1 - 2^{-n}$) we invert f_x on v. The final observarion is that, since $|S_x| \geq \frac{1}{2} \cdot 2^n$, the probability that none of $2n$ indepedently and uniformly selected n-bit strings hits S_x is very small (i.e., smaller than 2^{-n}). As in the proof of Proposition 3.1, the exponentially small error can be elliminated. It follows that the extractor strongly solve R_x under D_x. $\Box$

Acknowledgements

Work done while the first author was at the IBM T.J. Watson Research Center, New York.

References

[1] M. Bellare and O. Goldreich, "On Defining Proofs of Knowledge," *Advances in Cryptology – Crypto 92 Proceedings,* Lecture Notes in Computer Science Vol. 740, Springer-Verlag, E. Brickell, ed., 1992.

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

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

[4] U. Feige, and A. Shamir, "Witness Indistinguishability and Witness Hiding Protocols," *Proceedings of the Twenty Second Annual Symposium on the Theory of Computing,* ACM, 1990, pp 416-426.

[5] 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. (Preliminary version in the *17th STOC*, 1985.)

[6] 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 the *27th FOCS*, 1987, pp. 472-482.)

[7] M. Yung, "Zero-knowledge proofs of computational power," *Advances in Cryptology – Eurocrypt 89 Proceedings,* Lecture Notes in Computer Science Vol. 434, Springer-Verlag, J-J. Quisquater, J. Vandewille, ed., 1989.