Zero-Knowledge Proofs of Computational Power

Moti Yung
IBM Research Division
T.J. Watson Research Center
Yorktown Heights, NY 10598

(extended summary)

Abstract

Suppose that the NSA had announced the possession of an efficient factorization algorithm. The cryptology community, after recovering from the initial shock, would demand to see the algorithm and verify it. This request, however, could not be satisfied since the algorithm would probably be classified as top-secret information.

In this note we give a procedure which will satisfy both sides of the above imaginary dispute. This is a way in which one party can prove possession of some "computational power" (e.g., a special-purpose efficient factorization machine) without revealing any algorithmic detail about this computational task (e.g., the factoring algorithm).

1 Introduction

Interactive proofs were originally developed for membership in a language by Goldwasser, Micali and Rackoff [7]. In such a proof one party, the (P)rover, is engaged in an interactive protocol with another party, the (V)erifier. The task of the interaction is to convince the verifier (with overwhelming probability) that indeed the input belongs to some language.

The notion of "zero-knowledge" interactive proof [7] was introduced to capture the fact that an interaction validates the input membership, but does not give any extra advantage to the Verifier. See [7,6,8,3,11] for some of the examples of zero-knowledge interactive proofs. We assume familiarity with the above notions.

Later, interactive proofs "of knowledge" in which a polynomial-time P proves that it "knows" (possesses) a witness for some predicate about the input x was introduced. The formal specifications of such an interactive proof "of knowledge" are difficult to state precisely. Indeed, several such protocols had been proposed in the literature, and used in building cryptographic schemes, without a general

J.J. Quisquater and J. Vandewalle (Eds.): Advances in Cryptology - EUROCRYPT '89, LNCS 434, pp. 196-207, 1990.

definition of their properties. The formal definition and implementation of zero-knowledge interactive proof of knowledge was finally given by Feige, Fiat, and Shamir [5] and Tompa and Woll [13].

Suppose now that a polynomial-time P possesses — not just a witness— but access to some algorithm which gives some computational power. That is, it has some device (which can even be a machine TM to which P has only input-output access). The machine TM as a black box can solve instances of a given problem. We are interested in computations which are hard in the cryptographic sense. Thus, in the context of this paper an easy problem is assumed to be polynomial on the average, and a hard problem is almost always hard for problem size large enough. Here we suggest a way to formalize the notion of proving an access to such a computational device in a zero-knowledge fashion; we call the scheme "a zero-knowledge proof of computational power".

We have the following possible cryptographic applications in mind:

1. The scheme can serve as a method for revealing the discovery of a new crypt-analytic breakthrough in a responsible way. That is, convincing legal users of the broken system that it pays to change their system, but, on the other hand, doing so without publicly revealing the algorithmic details of the discovery. Thus, preventing exploitation of the new information by malicious users.

2. It can also serve consumers as a quality test for a cryptographic device (e.g., a decryption machine), while simultaneously protecting the dealer by preventing the potential customer from employing the device (e.g., using the test to decrypt messages he wants to decipher) during the test procedure.

Our scheme employs interactive proofs of knowledge [5,13,7,6,8], the perfect zero-knowledge interactive argument systems of Brassard, Chaum and Crépeau [3], and generation of random solved instances as defined by Hemachandra, Abadi, Allender, Broder and Feigenbaum [10].

2 From Proving Knowledge to Proving Power

While having a computational power (beyond, say, polynomial-time) is easy to express, the question we ask is how can we model a possession of computational power in the context of interaction between two untrusted parties. A computational power can be checked by revealing an algorithm, and proving its correctness

and running time. Doing so in a zero-knowledge fashion was formalized by Blum as proving any Theorem [2]. Such procedure, however requires that the prover has the verified algorithm, and cannot be employed when the prover has only input-output access to the algorithm. Therefore, we change the starting point and try to extend "proof of knowledge" to a proof which demonstrates more than just possession of a witness to some computation, but rather possession of "algorithmic power". We do not attempt to formalize computational power in general, but rather concentrate on the ability to solve problems which are hard in a cryptographic sense (on the average) as power.

2.1 Proof of Knowledge

Next we sketch the formalization of proving knowledge [5,13] which requires, that the polynomial-time P demonstrate the possession of a witness. This is done by forcing it to compute in a way which is tractable only if it has explicit access to the witness.

The two machines P and V share a common input x and the prover has to demonstrate that it possesses a witness w given to it as a private input. The witness is to the fact that $x \in Domain(R)$ for the relation R (i.e., w is such that $(x,w) \in R$). In the factorization example the relation is trivial, x is an integer, and w is its factors.

The interactive proof demonstrates the possession of w, through P's ability to answer questions posed by V, without actually revealing anything about the value of w. Thus, the "proof" is probabilistic in nature, and depends only on the messages sent by the two machines (in turns). The verifier is not able to read the prover's private input tape (which the prover is presumably using in order to compute its answers).

Formally, possession of a witness is modeled by means of a *recovery algorithm*, which is a Turing machine X (for "extractor") that interacts with any possible prover P^* in a special way — typically, while following a modified version of V's program. On input x, the goal of the recovery algorithm is to extract from P a witness. The machine X sends messages to and receives messages from P^*, just as V would do. The only difference is that while performing its computation, X may cause a machine configuration of P^* to be *saved*, and then, perhaps several turns later, X may restore the saved configuration of P^* and send different messages. The fact that X can backtrack the computation models the fact that in the "real"

interaction V may send a random message from a set of different ones and P^* should be able to answer them all. At the end of its interaction with P^*, X writes out its guess for w. The idea is that if X is able to extract the knowledge from the interaction when the messages produced by P are available to it then P^* "knows" the witness, and can, (if we modify it,) compute it.

Formally, let P denote the specified machine which has a private input (a witness) and follows the protocol, while P^* denote any machine acting as the legal P. For any input string x, the computation of the interaction denoted $(P, V)[x]$ either accepts or rejects. V's state at the end of the computation confirming the fact that P possesses a witness (accept) or not (reject).

We call a pair of interactive Turing machines (P, V) an *interactive proof-system of knowledge* for the binary relation R with error probability $\delta(k)$ (where k is the security parameter, i.e., the size of the input) if:

1. For any $(x, w) \in R$, if P possesses w then $(P, V)[x]$ accepts with (very high) probability $1 - \delta(k)$.

2. There exists a *recovery algorithm* X (associated with V), as described above, so that for any interactive Turing machine P^*, for any input string x, the probability of the event of both (1) $(P^*, V)[x]$ accepts, and (2) on input x, X interacting with P* outputs w which does not satisfy $(x, w) \in R$, is vanishingly small (less than $\delta(k)$).

The algorithm $X = X(V)$ works as specified, uniformly for any possible prover P^*, it has a limited access to the communication tapes only, and can backtrack the computation. This extractor works in expected polynomial time, thus the whole interaction of X with P^* is a feasible computation and can be run by P^* itself. This means that, with very high probability, if the interaction with V is an accepting one, then the machine acting as a prover can compute the witness; (if computing the witness is impossible for this machine, we may conclude that it was given to it).

An interaction like this is zero-knowledge if (loosely speaking) given a verifier V^* there is a machine which can compute its view of the interaction (which is a probability space) in expected polynomial time. Thus, its view of the computation does not provide any computational advantage beyond expected random polynomial time computation (in other words, provides "no knowledge").

2.2 Example: proving knowledge of factorization

Tompa and Woll [13] gave a zero-knowledge interactive proof of knowledge of factors of a given number. We start with their example and assume also that n is a composite with two large prime factors $n = pq$. Their proof relies on Rabin's proof of equivalence between extracting square roots and Factorization [12]. We also employ the zero-knowledge proof of residuosity of Goldwasser, Micali and Rackoff [7], formalized as proof of knowledge.

Assume P is given a quadratic residue (square), and is able to give a root. This implies, by Rabin equivalence, that P possesses the factors with probability at least $1/2$. If this is repeated k independent times and P is able to give a root each time then we conclude that P has the factors with probability greater than $1 - (1/2^k)$. In fact, because of the equivalence, an ability of any polynomial-time P^* to give a root means he has or can (with some probability) produce the factors. Namely, this is an interactive proof of knowledge.

The above scheme is not zero-knowledge and actually each time a root is revealed, also the factorization is revealed with probability $1/2$. This is a result of Rabin's equivalence (since P gives a square root, if it is a root not known to the verifier it might enable the factorization). Thus, instead of opening a root the prover will "prove" (in a zero-knowledge fashion) to the verifier that it "knows a root". This is still not a zero-knowledge proof, since the verifier may not choose a square according to the protocol requirements; an act which may give V an advantage. Thus, when it chooses and sends a square it is also required to prove the knowledge of a root— which implies that he indeed chose a square. This makes the protocol zero-knowledge (see [13]).

A high-level description of the protocol may look like this:

Protocol 1: Proving Possession of an Integer Factorization

Given n (which can be checked in random polynomial time not to be a prime power) as input.

For i=1,..,k do

1. The verifier chooses a quadratic residue y_i $(modn)$ and sends it
2. The verifier proves interactively that it knows a square root of y_i $(modn)$
3. The prover proves interactively that it knows a square root of y_i $(modn)$

If all proofs are successful V accepts, otherwise it rejects.

end-of-protocol

2.3 Towards a definition of proof of computational power

Our goal is to extend the "proof of knowledge" to "proof of computational power". In order to do this we must model what is a computational power. Our first observation is that the verifier may be simply modeled as interacting with another Turing machine TM whose "knowledge" of the witness is more complicated than its ability to simply read the bits of w from its private input tape. The value of an answer may be computed by TM in any possible way and we model it by letting the prover interact with a machine which gives answers to queries (say, gives an efficient factorization algorithm, or gives factors, or at least is able to answer queries specified by the verifier). This machine, TM, if restricted to polynomial time is assumed to possess the witness. The definition, however, is oblivious to the way TM computes, and an extractor X (in the definition of proof of knowledge) will produce the witness.

If we could just exhaustively feed the Prover (which has an access to the algorithm in its possession) with all possible inputs and observe that a correct result is given throughout, we could say that all possible witnesses "are possessed" by this prover. For example, asking about the factorizations of all numbers of a given size m, enables us to conclude that this algorithm has the claimed power.

Obviously, however, the number of inputs is prohibitively large for such an interaction which is required to end in polynomial time. Thus we have to relax the interactive proof and turn it into a "statistical test" which samples a random set of inputs and observe how the prover reacts to them.

A statistical test can be based on a law of large numbers. Let $F_k(E)$ be the frequency of the test of k independent trials of event E. For a sample size $k \geq 1$, applying (for example) Bernstein's strong version of the law of large numbers, assuming the actual probability of the parameter is p, then for each $\delta \leq p(1-p)$ the following holds: $prob\{|F_k(E) - p| \geq \delta\} \leq 2e^{-k/(p(1-p))^2}$. This gives a statistical test (by choosing a large enough k) which makes us accept the hypothesis that the probability is indeed p with negligible error. Actually we are interested in the one-sided hypothesis that the actual probability is (almost) one, and we will reject the alternative hypothesis that $p = r$ for some constant r based on the fact that $F_k(E) - r > \delta$.

Next, we start with the factoring example and then the formal definitions.

3 Proving the Power to Factor Integers

Assume that the prover claims that he can factor numbers of a certain size (say, m-bit long numbers). This gives us a probability space which is samplable [1] (i.e., a number with its factorization can be generated at random). We say that a party possesses the ability to factor if he can factor any number (or all but a negligible fraction of the numbers) in the space. While, on the other hand, if he does not have this power we assume that factoring is $(1-\epsilon)-hard$ for him, that is, he cannot factor more then ϵ fraction of the factors (we call ϵ the tractability fraction).

The intractability assumption of factoring (of composites which are multiplication of two large primes of equal size) is that it is hard on the average. Formally, it says that for any polynomial R, the tractability fraction $\epsilon \leq 1/R(m)$ for integers of all size parameter m large enough. Thus, in particular, we can assume that the tractability fraction ϵ is smaller than $1/2$ (by the intractability assumption, this is true from a certain size m and on).

The nature of a problem being hard on the average then, suggests the following process. Let V sample and check P's ability to factor. V chooses a large enough number of integers uniformly at random from the sample space and P provides factors to all of them. If P possesses TM and can factor, then he should provide factorization to all the sample. On the other hand, for any polynomial-time P*, eventually its probability of factoring is less than $1/2$ for a random choice of an integer. Thus, this constitutes a statistical test.

Let $F_k(E)$ be the frequency of the test of k independent trials of the event that a number of size m (large enough) is being factored by a prover. Assume the probability of factoring is (smaller than or) equal $1/2$. (Recall that by the intractability assumption, this is true for large enough m). For a sample size $k \geq 1$, applying Bernstein's law (we are interested in a one-sided version of it) we get that $prob\{F_k(E) \geq 3/4\} \leq e^{-k/16}$. This gives a polynomial sample size k which with negligible probability will give a successful statistical proof (of the ability to factor all numbers in the given sample space). A polynomial time machine not augmented by TM will fail with overwhelming probability to produce $3/4$ of the requests and the assumption that $p = 1/2$ will be rejected. Thus, a successful test implies that the prover has some computational power and it can factor most of the numbers (which is, by the intractability of factoring, beyond average-P). In addition, the true prover possessing TM, will be able to answer all queries with

overwhelming probability, since TM can factor (almost) all numbers in the space.

Again, implementing this suggested statistical test as a protocol is not zero-knowledge, however, the following refinement of it is (by the fact that it uses protocol 1 which is zero-knowledge).

Protocol 2: Proving Power to Factor numbers

On input m size parameter, for i=1,..,k do:

1. V picks a random integer n_i of size m and sends it
2. V uses protocol 1 to prove to P that it knows the factorization of n_i
3. P uses protocol 1 to prove to V that it knows the factorization of n_i

If all proofs are successful V accepts, otherwise it rejects.

end-of-protocol

This is a statistical test, which (as we claimed above) proves that P has the power as claimed. It is also (perfectly) zero-knowledge— given a verifier V^* the space of possible interactions with P can be exactly simulated (or if can factor most of the numbers it can be almost simulated with a negligible error).

4 Interactive Proof of Computational Power

In this section we generalize the above interaction to a more general class of problems which are assumed to be hard on the average; we define the class of problems to which such a proof can be applied.

4.1 Definitions

Let PR be a $(1-\epsilon)-hard$ problem (as defined above) with size parameter m and let it be a samplable problem: denote the sample space $S(m)$. That is, a problem in NP for which it is possible to generate random instances. It can be treated as a collection of relations $\{R(x,w)|x \in S(m), m \in I \subseteq N\}$.

A problem as above is hard on the average and a statistical test can "measure" the difference between a polynomial time machine and a machine with augmented power.

When a witness (a solution) is generated together with the instance, a samplable problem is exactly a problem which has a $(1-\epsilon)-invulnerable$ generation as defined by Hemachandra, Abadi, Allender, Broder and Feigenbaum [10]. This

is, generating solved instances which cannot be solved in polynomial-time with some fixed probability $(1 - \epsilon)$.

For our purposes, the requirement of random generation can be weakened. We need not generate in polynomial time a witness, but rather we should be able to generate in polynomial time a random YES-instance with the ability to interactively prove (by a polynomial-time machine, our verifier) that it is indeed an instance which was sampled correctly at random. We call such a problem a *samplable verifiable problem.*

Next we define the interactive proof of computational power. For any input string m, the computation $(P, V)[m]$ either accepts or rejects as before. P is the specified polynomial-time machine with the power to solve instances (it possesses a machine TM with this power), while P^* is any polynomial-time machine.

Let X be a procedure which can interact with the prover P and can backtrack the computation, furthermore, X is given an input from an input machine U which generates random instances; these instances are *unsolved* for X (namely, X does not "know" the witnesses); X tries to use the prover to solve these instances (via the interaction).

We call a pair of interactive Turing machines (P, V) an *interactive proof-system of computational power* for the above problem PR with error probability $\delta(m)$ (where m is the security parameter) if for m large enough:

1. If P is the specified prover which possesses a machine TM with the ability to solve (almost) all instances from $S(m)$, then V accepts (with overwhelming probability $1 - \delta(m)$).

2. For any machine P^* acting as a prover, the probability that $(P^*, V)[x]$ accepts and that an extraction procedure X which samples unsolved instances given by U, using (interacting with) P^*, and does not output the witnesses of all sampled instances is vanishingly small (less than $\delta(m)$).

The algorithm X works as specified, uniformly for any possible prover P*, it has a limited access to the communication tapes only, and can backtrack the computation. This extractor works in expected polynomial time, thus the whole interaction of X with P* is a feasible computation (and can be run by P^* itself). This means again that, with very high probability, if the interaction with V is an accepting one, then the machine acting as a prover can compute all the witnesses as required and therefore can pass the statistical test (which means it has some

computational power, with respect to the hard problem in question). Note that we use the fact that the instances produced by U are random and the solved instances generated by V are sampled from the same (or almost the same) distribution, therefore they both should fail and succeed with (almost) the same chance.

Now we can state the following

Theorem 1 *Under the intractability assumption of factoring (of large numbers multiplication of two large primes), protocol 2 above is a "perfect zero-knowledge proof system of the computational power to factor".*

4.2 A Protocol Scheme

The following is a general scheme for a samplable (verifiable) problem PR. Recall that an instance of PR is in the domain of a relation R (it has a witness) and the polynomial-time verifier can prove in zero-knowledge the validity of the chosen instance, while the prover has to be able to prove the knowledge of a witness.

Protocol 3: Proving Computational Power for Samplable Verifiable Problem

On input m (size parameter), for i=1,..,k do (k is a function of the required statistical confidence δ):

1. V picks a random solved instance n_i of size m and sends it
2. V uses a perfectly zero-knowledge proof that n_i is in the domain of the relation R
3. P proves in perfect zero-knowledge that it knows a witness w_i such that $R(n_i, w_i)$.

If all proofs are successful V accepts, otherwise he rejects.

end-of-protocol

Theorem 2 *The above protocol scheme is a "(perfect) [computational] zero-knowledge (argument) [proof] system of computational power" for samplable verifiable problems which are* $(1-\epsilon)-$ *hard.*

The above class of PR problems includes factoring as well as other problems used as candidates for cryptographic applications (Discrete Logarithm, RSA inversion, etc.). These problems have this property with direct verifiability in perfect zero-knowledge, (see [10]). They are assumed to be hard on the average and

an overwhelming success in solving random instances of them for large enough security parameter, implies a computational power beyond P. (We remark that proving of computational power can be extended to a larger set of problems, but here we are interested in PR as the set most relevant to applications). The zero-knowledge property of the protocol is derived from the zero-knowledge property of proving knowledge. Note that the exact length and confidence parameters of the proofs in steps 2 and 3 are determined by the confidence parameter δ.

When we allow computational zero-knowledge we can allow proving invulnerable NP-problems in general (an example of such problem of a special size Clique has been recently given by Gurevich and Shelah [9]); this can be done under additional cryptographic assumption.

Next we suggest what to do when the interactive proofs cannot be implemented in perfect zero-knowledge without additional assumption, and an assumption is needed.

The proof of the correct sampling in step 2, is given by the verifier to a prover whose power is not known. In particular, P may hold some advantage in factorization, and our encryption on which the security (zero-knowledge-ness) of the proof is accepted may rely on the same problem (that is, factorization)! In this case encrypting the proof may not be enough, hiding it perfectly is required. Therefore, a natural proof system for an NP-statement to be used in this context is Brassard, Chaum, Crépeau's system of perfect zero-knowledge interactive argument proof [3]. The correctness of the proof relies on V's restriction to polynomial time (which means it can cheat with negligible probability).

In step 3, when P proves to V, if this step can be done perfectly the proof system is perfectly zero-knowledge (otherwise, it is computationally zero-knowledge). Actually, we observe that P can prove using an argument system as well (making the system perfectly zero-knowledge). The proof system is correct based on the underlying cryptographic assumption. If the legal P (possessing the extra power) interacts with V it will follow the protocol, and will not abuse its power. On the other hand, any cheating P^* (which is in polynomial time) will not be able to cheat in the proof under the cryptographic assumption, but with a negligible probability. This completes the proof. The interactive argument system originated in [3] can be implemented under a general assumption that one-way homomorphism exists [11], and it requires only a constant number of iterations [4] which is important for various applications.

References

[1] E. Bach, *Generating Random Numbers with Known Factors*, SIAM Journal on Computing.

[2] M. Blum, *How to Prove a Theorem so No One can Claim It*, International Conf. of. Math., 1987.

[3] G. Brassard, D. Chaum and Crépeau C., *Minimum Disclosure Proofs of Knowledge*, J. Comp. Sys. Sci. 37-2, pp. 156-189.

[4] G. Brassard, Crpeau C. and M. Yung, *Any NP statement can be proved in perfect zero-knowledge in bounded number of rounds* , ICALP 1989.

[5] U. Feige, A. Fiat and A. Shamir, *Zero-Knowledge Proof of Identity*, Proc. 19th STOC, 1987, pp. 210-217.

[6] Z. Galil,, S. Haber and M. Yung, *A Private Interactive Test of a Boolean Predicate and Minimum-Knowledge Public-Key Cryptosystems*, FOCS, 1985 pp. 360-371.

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

[8] S. Goldreich, S. Micali and A. Wigderson, *Proofs that Yields Nothing But their Validity, and a Methodology of Cryptographic Protocol Design*, Proc. 27th FOCS, 1986.

[9] U. Gurevich and S. Shelah, , Private Communication.

[10] L. Hemachandra, M. Abadi, E. Allender, A. Broder, and J. Feigenbaum *On Generating Solved Instances of Computational Problems* , Proc. of Crypto 88.

[11] R. Impagliazzo and M. Yung, *Direct Minimum-Knowledge Computations* , Crypto 87.

[12] M. O. Rabin, *Digital Signatures and Public Key Functions as Intractable as Factoring*, Technical Memo TM-212, Lab. for Computer Science, MIT, 1979.

[13] M. Tompa and H. Woll, *Random Self-reducibility and Zero-Knowledge Interactive Proofs of Possession of Information*, FOCS, 1987, pp 472-482.