Computationally convincing proofs of knowledge

(extended abstract)

Gilles Brassard [*]
Université de Montréal [†]

Claude Crépeau [‡]
Université de Paris–Sud [§]

Sophie Laplante [¶]
Université de Montréal [†]

Christian Léger [‖]
Université de Montréal [†]

November 3, 1990

1 Introduction

The original definition of interactive proof–systems, as given by Goldwasser, Micali and Rackoff, does not impose limits on the prover's computing power [GMR].[1] Hence, the prover may wish to convince the verifier of the truth of an assertion, but it does not really make sense for the prover to convince the verifier that she[2] *knows* a proof of the assertion (of course she knows such a proof if it exists since she is all–powerful). Nevertheless, Feige, Fiat and Shamir [FFS], and Tompa and Woll [TW] have given formal definitions of what should constitute a proof of knowledge in the context of interactive proof–systems.

Brassard, Chaum and Crépeau have investigated a different setting, in which the prover's computing power is limited [BC1,C,BCC]. The resulting protocols are convincing for the verifier provided that he believes that the prover cannot break a given cryptographic assumption while the protocol is in progress. Consequently, such protocols are merely *computationally convincing*, as opposed to Goldwasser–Micali–Rackoff's *statis-*

[*]Supported in part by Canada NSERC Grant A4107.

[†]Département d'informatique et de recherche opérationnelle, Université de Montréal, C.P. 6128, succ. "A", Montréal, Québec, Canada H3C 3J7.

[‡]Supported in part by NSERC Postgraduate and Postdoctorate Scholarships.

[§]Laboratoire de Recherche Informatique, Université de Paris–Sud, Bâtiment 490, 91405 Orsay, France. This research was performed in part while the author was a graduate student at M.I.T.

[¶]Supported in part by an NSERC Postgraduate Scholarship.

[‖]Supported in part by NSERC and by FCAR.

[1] For the purpose of this extended abstract, we assume that the reader is familiar with the notions of proof–systems and of zero-knowledge [GMR,BCC].

[2] For convenience, we refer to the prover as a "she", and to the verifier as a "he", whereas both the simulator and the extractor – see Section 3 – are referred to as "it"s.

tically convincing proof–systems (this terminology was recently suggested by Chaum). Computationally convincing protocols are usually referred to as *arguments* [BC2]. The advantage of the apparently weaker notion of argument over that of proof–systems is that arguments can be *perfect* zero-knowledge [GMW] for **NP**–complete problems, as shown in [BCC], which is ruled out for proof–systems by a result of Fortnow (unless the polynomial–time hierarchy collapses) [F].

The natural question of defining the notion of "computationally convincing proof of knowledge" was addressed initially by Boyar, Lund and Peralta [BLP]. In this paper, we give a more general definition, which is capable of taking into account very adversarial behaviour from the prover. We also prove that constant-round arguments for **NP**–complete problems are proofs of knowledge, which is much more difficult to prove than for k-round protocols. Finally, we exhibit a curious protocol that forces us to question the full generality of even our definition of "computationally convincing": it may be that a more general definition is needed still, and that we do not yet know precisely when we should be convinced that a protocol is convincing.

2 Honest, cheating, well-behaved and reluctant provers

When an interactive protocol between a prover and a verifier is specified [GMR], the *honest* behaviour of both parties is determined exactly (modulo the fact that such behaviour can be defined in function of random choices). Any party that deviates from his of her prescribed behaviour is known as a *cheater*. In some cases, a cheater produces an interaction identical to the interaction that the honest party would produce. This can occur, for instance, if the honest party is expected to choose a random i, compute $x = \alpha^i \bmod p$, and transmit x to the other party (where α is a generator of $\mathbf{Z}_p^*$), whereas the cheating party computes x as $2\alpha^i \bmod p$ for a randomly chosen i. This begs the question of when should a protocol be convincing despite the possibility of undetectable cheating behaviour on the prover's side. In other cases, which are easier to deal with, a cheater deviates from the prescribed protocol in a way that can be detected by the other party, which will usually cause the protocol to be terminated by the party detecting such a deviation.

A *well-behaved* prover is a prover that acts in such a way that she will never be called a cheater when interacting with an honest verifier (except perhaps with negligible probability if such is the case already for the honest prover). In particular, honest provers are well-behaved. In the context of arguments, it can be very interesting for a cheating prover to be well-behaved, since this allows her to escape detection. For instance, assume that she can break the cryptographic assumption upon which the verifier's confidence is based. This allows her to change her commitments at will and thus be well-behaved in convincing the verifier that she knows a satisfying assignment for a given *non*-satisfiable Boolean expression. (Notice that in the context of proof–systems à la Goldwasser, Micali and Rackoff, however, a prover would have no possible advantage in being merely well-behaved rather than completely honest.)

Both in proof–systems and in arguments, it is always possible that a very lucky cheating prover will not be caught by the honest verifier just because she managed to guess ahead of time all the challenges that the verifier will issue. This probability of lucky undetected cheating can either be required to be superpolynomially small as a function of the size of the instance considered, or exponentially small as a function of an explicit so-called *confidence parameter* (depending on the author's preference). We favour the second approach, in which the prover and verifier agree beforehand on the value of an integer k such that the verifier is willing to tolerate a probability 2^{-k} that the prover, if dishonest, will be lucky enough not be caught. We say that the prover is *reluctant* if she has a non-negligible probability of being caught cheating by the verifier, yet her probability of success is strictly better than 2^{-k}.

In the context of proof–systems, there is no possible advantage for a prover to be reluctant. Indeed, if she can achieve a probability of success better than 2^{-k}, it is necessarily because the statement she claims is in fact true, and therefore she could just as well be honest and make sure she will not be caught (except perhaps with negligible probability) [3]. Here again, however, the situation is different with arguments. It makes perfect sense for a cheating prover to try to break the cryptographic assumption while the protocol is in progress. Such a prover could succeed some of the times but not always. As a result, she could be caught cheating with a non-negligible probability, yet she could also succeed with a probability significantly greater than 2^{-k}. If the reluctant prover's probability of undetected cheating is $2^{-k} + \wp$ for $\wp > 0$, we say that her *advantage* is $\wp$.

3 The extractor

The notion of *simulator* was already put forward in the very first paper to describe the concept of zero-knowledge [GMR]. Recall that the simulator's purpose is to efficiently produce a view of what the verifier would see if he were to interact with the prover. The existence of this simulator demonstrates clearly that the verifier did not need the prover to get whatever he obtained, and thus that he learned nothing from the prover. Usually, the simulator achieves this goal by using the verifier as a black-box on which it has complete control. In particular, the simulator controls the verifier's coin tosses, it can take snapshots of the verifier, and it can restore the verifier to a previous state. Also, the simulator can talk to the verifier, making him believe that he is talking to the prover.

Similarly, it is natural to consider yet a fourth actor (introduced implicitly by Tompa and Woll [TW], explicitly but namelessly by Feige, Fiat and Shamir [FFS], and named *observer* by Boyar, Lund and Peralta [BLP]). We prefer to call this actor the *extractor* because we give it a more active role than Boyar, Lund and Peralta. Just as the

[3] Nevertheless, the term "reluctant" was coined thinking of an infinitely powerful prover that would occasionally and purposely misbehave in such a way as to get caught for some strange reason.

simulator is given complete control over the verifier, the extractor is given complete control over the prover. In particular, the extractor can talk to the prover, making her believe that she is talking to the verifier. Here again, the extractor can take snapshots of the prover and restore her later to a previous state. The extractor's purpose is to extract the "secret knowledge" that the prover claims to have. Thus, the existence of an efficient extractor demonstrates that the prover was honest in claiming her knowledge. The extractor is the main tool that has been proposed so far to formalize the notion of "proof of knowledge".

For instance, assume that the prover claims that some Boolean formula Ψ is satisfiable. For simplicity, let us assume initially that the prover is well-behaved. If the interactive protocol under consideration is a proof–system à la Goldwasser, Micali and Rackoff, as considered by Feige, Fiat and Shamir and by Tompa and Woll, the extractor's goal would be to obtain a satisfying assignment for Ψ. This definition of what an extractor should do is fine in the context of a proof–system. However, it would obviously be inappropriate if the protocol under consideration is merely computationally convincing, because Ψ could in fact be non-satisfiable even if the prover is well-behaved — but of course not honest. In order to take this possibility into account, Boyar, Lund and Peralta proposed that the extractor's purpose (or rather the observer's) should be *either* to obtain a satisfying assignment for Ψ, or to break the protocol's bit commitment scheme [BLP]. Here, we give a different and slightly more general definition, which is not dependent upon the notion of bit commitment, and which can take into account reluctant provers.

Let us say that the cryptographic assumption needed for the verifier to be convinced by the protocol is that the prover cannot perform task $\mathcal{T}(x)$ while the protocol is in progress, except with negligible probability, where x is an instance of $\mathcal{T}$ normally chosen by the verifier. For example, $\mathcal{T}(x)$ could be the task of extracting the discrete logarithm of x. When dealing with a well-behaved prover, the extractor's task is either to obtain a satisfying assignment for Ψ *or to succeed at performing task* $\mathcal{T}(x)$ on an arbitrary (**not** necessarily random) instance x of $\mathcal{T}$ imposed upon the extractor from the outside. The extractor must achieve this goal efficiently, not counting the time spent waiting for the prover to perform her share of the protocol. Intuitively, such an extractor shows that either the prover knows a satisfying assignment for Ψ (or can compute one easily from what she knows), or that she has the ability of breaking the cryptographic assumption upon which the protocol is based.

But what if the prover is reluctant? Let $\wp$ be the prover's advantage. (Recall that this means that the prover has probability $2^{-k} + \wp$ of not being caught cheating when interacting with the honest verifier, where k is the protocol's confidence parameter). In this case, we still ask that the extractor should succeed at either figuring out a satisfying assignment for Ψ or performing task $\mathcal{T}(x)$, but it is allowed to take $poly(1/\wp)$ times longer to achieve this goal than if the prover had been well-behaved, where *poly* is a reasonably small polynomial (ideally a linear polynomial).

To summarize the proposed definition, a protocol is a *computationally convincing proof of knowledge* based on cryptographic task $\mathcal{T}$ with confidence parameter k if there exists an extractor capable of dealing with any prover whose probability of not getting caught cheating by the honest verifier is $2^{-k} + \wp$ for a strictly positive $\wp$. Assume that the prover claims to know a satisfying assignment for Ψ. The extractor is given an arbitrary instance x of task $\mathcal{T}$ and complete control over the prover. The extractor's goal is either to figure out a satisfying assignment for Ψ, or to solve task $\mathcal{T}(x)$. The extractor's expected time must be polynomial in $1/\wp$ and in the expected time that the protocol between this prover and the honest verifier would have taken.

4 Examples

It is rather easy to prove that the early perfect zero-knowledge arguments for **NP**–complete problems given by Brassard, Chaum and Crépeau [BCC] are proofs of knowledge, as pointed out in [BLP]. In order to obtain more challenging examples of natural protocols that deserve to be shown to be proofs of knowledge, one must consider more recent protocols. For instance, [BLP] considered the subtle case of noninteractive treatment of parity gates, which leads to surprising difficulties.

In this paper, we consider the case of *constant-round* protocols, that is protocols that can achieve an arbitrarily high confidence parameter k with a fixed number of rounds of interaction between the prover and the verifier. These protocols are harder to deal with because the extractor's main tool in all previous proofs of knowledge (computationally convincing or otherwise) has been to get the prover to answer two different sets of challenges on the same prover's set of commitments. The difficulty with constant-round protocols is that they require the verifier (hence the extractor) to commit to her (its) challenges early in the protocol, i.e. before getting to see the prover's commitments. This makes it possible for the prover to choose her commitments as a function of the verifier's (or the extractor's) committed challenges.

4.1 The protocol of [BCY]

The first perfect zero-knowledge constant-round argument ever proposed for an **NP**–complete problem is due to Brassard, Crépeau and Yung [BCY]. Unfortunately, it is rather complicated, and until recently it was conjectured not to be a computationally convincing proof of knowledge according to the above definition. Here, we very briefly sketch a proof that this conjecture was false. In order to simplify, we will assume that the reader is familiar with the protocol of [BCY] [4] and we will only consider the case of well-behaved provers (a detailed description of the general extractor and a proof that it works even with reluctant provers has been worked out, but it would require more pages than allowed in this extended abstract).

Let p be a large prime and α be a certified generator of $\mathbf{Z}_p^*$ that the prover and

[4] We apologize to others, but summarizing this protocol here would take us much too far. This subsection can be skipped without loss of continuity.

the verifier agree to use. Let the cryptographic assumption be that it is infeasible to extract discrete logarithms modulo p. Let $s \in \mathbf{Z}_p^*$ be provided "from the outside" to the extractor. Let Ψ be a Boolean expression for which the prover claims to know a satisfying assignment. The extractor's purpose will be either to obtain a satisfying assignment for Ψ, or to extract the discrete logarithm of s. Playing the role of the honest verifier (but using the externally supplied s instead of choosing one at random), the extractor follows honestly the protocol with the prover until the step at which she replies to the challenges given about her control blobs. At this point, the extractor resets the prover to her state just before accepting these challenges, and the extractor gives her the complementary set of challenges. From the prover's reply to these two sets of challenges, the extractor can easily determine a way in which to open each and every of the actual blobs initially supplied by the prover. For each i, $1 \leq i \leq k$, the extractor decides whether this way of opening the blobs that hid the ith "scrambled circuit" would have been an acceptable answer to challenge 0. (Note that this way of opening the blobs could provide a bad answer to challenge 0 even if the prover is assumed to be well-behaved because, if *really* faced with challenge 0, perhaps the prover would open the blobs differently.) The extractor continues with the protocol honestly by opening its main challenges in the way that it had committed to very early in the protocol. The extractor collects the prover's answers to them. With probability $1 - 2^{-k}$ (because the prover is assumed to be well-behaved), there is at least one i such that one of two good things happens, in which case the extractor will have succeeded:

- The extractor had deduced earlier a way of opening the actual blobs for circuit i that would *not* have satisfied challenge 0, yet challenge 0 was asked of the prover for this value of i and she gave a valid answer. In this case, the extractor has seen how to open at least one of the blobs in two different ways, and this is enough to deduce efficiently the discrete logarithm of s.

- The extractor had deduced earlier a way of opening the actual blobs for circuit i that *would* have satisfied challenge 0. Challenge 1 was asked of the prover for this value of i and she gave a valid answer. In this case, the extractor has seen a valid answer to both challenges for the same value of i. Either these two answers are compatible in the sense that whenever a given blob is opened by the prover under both challenges it is always opened to show the same bit, or not. If the answers are compatible, the extractor can easily deduce a satisfying assignment for Ψ. If not, the extractor has seen how to open at least one of the blobs in two different ways, and this is enough to deduce efficiently the discrete logarithm of s.

4.2 The protocol of [B]

A new constant-round argument for **NP**–complete problems was recently discovered by Brassard [B]. It is easier to understand than the [BCC] protocol *and* it is easier to prove that it is a proof of knowledge [5]. In this subsection, we give in detail an

[5] On the other hand, it is much harder to prove that it is perfect zero-knowledge, which does not concern us here.

extractor capable of dealing with well-behaved provers for this protocol. We then sketch a modification that allows the extractor to deal also with reluctant provers. Finally, a small modification to this protocol creates a very intriguing phenomenon: it yields a protocol that remains computationally convincing, yet that seems no longer to be a proof of knowledge according to our definition.

First of all, let us lift from [B] the relevant protocol. In order to understand this protocol, it suffices to know that x_i is a bit string that the prover will commit to, waiting for a challenge y_i. If the challenge is 0, the prover should open all the bits of x_i. If the challenge is 1, the prover should selectively open some of the bits of x_i. This is done in such a way that it is easy to deduce a satisfying assignment for Ψ from the answer to both challenges (provided that the commitments that are opened according to both challenges are opened consistently), but that seeing the answer to only one challenge provides no information. In the protocol below, z_i stands for the information that the prover supplies in order to meet challenge y_i. All the arithmetic in the protocol takes place in $\mathbf{Z}_p^*$, hence it is implicitly performed modulo p.

Protocol 1 (the protocol of [B])

1: Initialization: The prover and verifier agree on a large prime p for which they both know the factorization of $p-1$, and they agree on a generator α of $\mathbf{Z}_p^*$. They also agree on a confidence parameter k.

2: The prover chooses a random $\ell \in \mathbf{Z}_{p-1}$;
she computes $t = \alpha^\ell$;
she sends t to the verifier.

3: The verifier chooses a random $s \in \mathbf{Z}_p^*$, random bits $y_1, y_2, \ldots, y_k$,
and random $w_1, w_2, \ldots, w_k$ in $\mathbf{Z}_{p-1}$;
for each i, $1 \leq i \leq k$, he computes $h_i = \alpha^{w_i} t^{y_i}$;
the verifier sends s and $h_1, h_2, \ldots, h_k$ to the prover.

4: The prover commits herself to $x_1, x_2, \ldots, x_k$. All the prover's commitments are made with respect to s, i.e. whenever the prover wishes to commit to bit b, she does so by computing $\alpha^r s^b$ for a random $r \in \mathbf{Z}_{p-1}$.

5: The verifier opens challenges $y_1, y_2, \ldots, y_k$ for the prover by revealing $w_1, w_2, \ldots, w_k$.

6: The prover verifies that $h_i = \alpha^{w_i} t^{y_i}$ for all i's.
If not, she terminates the protocol;
if so, she meets the challenges by sending $z_1, z_2, \ldots, z_k$ to the verifier.
Also, she gives ℓ to the verifier.

7: The verifier accepts if and only if $t = \alpha^\ell$ and the prover was able to meet each and every one of the challenges.

In order to prove that this protocol is a computationally convincing proof of knowledge, let us initially consider the case of a well-behaved prover. Let Ψ be the boolean expression for which the prover claims to know a satisfying assignment. The extractor proceeds as follows, where "step i" refers to Protocol 1, whereas "step Ei" refers to Protocol 2.

Protocol 2 (an extractor for Protocol 1)

E1: The extractor follows step 1 honestly with the prover. This results in a choice of p, α and k.

E2: The extractor waits for the prover to carry out step 2 and supply t.

E3: The extractor gets s from the outside. It chooses the y_i's and the w_i's, and computes the h_i's as the honest verifier would at step 3. It sends s and the h_i's to the prover.

E4: The extractor waits for the prover to carry out step 4 and supply her commitments to her secret x_i's.

E5: The extractor takes a snapshot $\mathcal{S}5$ of the prover. The extractor opens its challenges honestly, as the honest verifier would at step 5.

E6: The extractor waits for the prover to carry out step 6 and supply the z_i's and ℓ.

E7: The extractor restores the prover to snapshot state $\mathcal{S}5$.

E8: Thanks to its knowledge of ℓ, kindly supplied by the prover at step E6, the extractor is now in a position to cheat the "commitments" that it had offered at step E3. The extractor uses this information to carry out step 5 again, but with a different collection of randomly chosen challenges $\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_k$.

E9: The extractor waits for the prover to carry out step 6 again and supply the corresponding new collection of $\hat{z}_i$'s.

E10: Consider any i such that $y_i \neq \hat{y}_i$. Either all the commitments opened by the prover about the bits of x_i are opened consistently in z_i and $\hat{z}_i$ or not.

- If this is so, the extractor has just seen consistent and valid (because by assumption the prover is well-behaved) answers to both challenges about the same x_i. From this, the extractor can easily deduce a satisfying assignment for Ψ.
- Otherwise, the extractor has seen at least one commitment opened by the prover in two different ways. From this, the extractor can easily deduce the discrete logarithm of s since the prover's commitments were based on s.

It is clear that this extractor succeeds with certainty if it deals with a well-behaved prover. Moreover, it does not take more than twice the time needed to carry out the actual protocol between this prover and the honest verifier. But it is just as clear that it could fail miserably against a reluctant prover. In fact, it is easy to design a reluctant prover that will make this extractor fail *with certainty* for specific values of s.

Let us now sketch how to deal with reluctant provers. Let k be the confidence parameter. Consider a reluctant prover whose probability of undetected cheating is $2^{-k} + \wp$ when interacting with the honest verifier. A first idea would be to design an extractor that would follow Protocol 2 exactly, but would start all over again whenever the prover misbehaves. A careless analysis could make us believe that this extractor would have to try on average less than $1/\wp$ times before being lucky enough to observe a run in which the prover does not misbehave. Unfortunately, this is not so for two different reasons. First of all, it could be that the prover will *always* misbehave if given some specific s at step 3 (the probability that the prover does not misbehave is taken over all possible random choices of both parties, and this includes the random choice of s that the verifier is required to make at step 3). In this case, the extractor would be caught into an infinite loop. The other problem is that the extractor needs not one, but two correlated runs of the protocol for which the prover does not misbehave. The fact that these required runs are correlated (they share steps 1 through 4) introduces additional difficulties because the probability that the prover will not misbehave once a *specific* run of protocol 1 reaches step 5 could depend on the specific choices of s and $h_1, h_2, \ldots, h_k$. However, this probability cannot depend on the specific choice of $y_1, y_2, \ldots, y_k$ because these are information-theoretically hidden from the prover at that point.

It is easy to deal with the first difficulty: At step E3 the extractor calls $\hat{s}$ what it gets from the outside, it chooses a random $j \in \mathbf{Z}_{p-1}$, and it computes $s = \hat{s}\alpha^j$ (in $\mathbf{Z}_p^*$, of course). Now s is a random element of $\mathbf{Z}_p^*$ but the extractor can easily compute the discrete logarithm of the prescribed $\hat{s}$ once and if the discrete logarithm of s is obtained.

In order to deal with the second difficulty, a careful analysis of the probabilities involved is necessary. The extractor follows step E1 through step E6 (with the small change already indicated for step E3), except that it starts all over again whenever the prover misbehaves. Assuming that the prover has not yet misbehaved and that step E6 is completed successfully, the extractor tries to carry out step E7 to step E9. If the prover misbehaves, the extractor does **not** go back to the starting point. Rather, it tries again step E7 to step E9. The extractor tries again and again these steps until a successful round is obtained, *but with a time limit*. After more than $1/\wp$ attempts have failed, the extractor gives up and goes back to the starting point. If, on the other hand, an execution of step E7 to step E9 succeeds, the extractor can proceed to step E10 and complete its task.

A detailed analysis of this strategy shows that the total expected number of times that the extractor has to restore the prover (including going back to the starting point) is in the order of $1/\wp$. This analysis will be provided in the final paper. (It is easy to show that this number is in the order of $1/\wp^2$, but in fact it is in the order of $1/\wp$.) As a consequence, our extractor achieves its goal in expected time that is simply $c/\wp$

times greater than the time taken by the actual protocol between this prover and the honest verifier, for a (small) constant c. This completes the proof that Protocol 1 is a computationally convincing proof of knowledge.

A variation on this extractor works even if the extractor does not know $\wp$ beforehand. The idea is that the prover can figure out an estimate on $\wp$ from the number of unsuccessful tries before the first time that it manages to reach step E7. Details will be given in the final paper.

5 An intriguing question

Consider a slight modification in Protocol 1, in which we no longer require the prover to supply ℓ at step 6. It seems clear that the resulting protocol is just as safe from the verifier's point of view. This is so because the only way the prover could use her additional freedom towards cheating would be by choosing t at step 2 in a way that she does not know its discrete logarithm (because if she actually does know it, she has nothing to lose by revealing it at the very end of the protocol). But this choice of t is done so early in the protocol that it cannot help the prover cheat no matter how cleverly she chooses it.

Nevertheless, the modified protocol does not appear to be a proof of knowledge even against well-behaved provers. The problem is that a well-behaved (yet cheating) prover could make $x_1, x_2, \ldots, x_k$, as well as the way in which she commits to their bits, depend on s and $h_1, h_2, \ldots, h_k$. This, together with the extractor's ignorance of the discrete logarithm of t (whose knowledge is needed to cheat its commitments), would deprive the extractor of its chances of getting the prover to show two compatible runs of Protocol 1. Therefore, this modified protocol could be the first interactive protocol ever proposed that is computationally convincing, yet that is not a proof of knowledge according to our definition. We are currently working out a simpler example of such behaviour, which would indicate the same problem with the original definition of proof of knowledge given by Feige, Fiat and Shamir [FFS].

References

[BLP] Boyar, J., Lund, C. and Peralta, R., "On the communication complexity of zero-knowledge proofs", manuscript submitted to the *Journal of Cryptology*, 1989.

[B] Brassard, G., "Constant-round perfect zero-knowledge made easy (and efficient)", manuscript available from the author, 1990.

[BCC] Brassard, G., Chaum, D. and Crépeau, C., "Minimum disclosure proofs of knowledge", *Journal of Computer and System Sciences*, Vol. 37, no. 2, 1988, pp. 156 – 189.

[BC1] Brassard, G. and Crépeau, C., "Non-transitive transfer of confidence: A *perfect* zero-knowledge interactive protocol for SAT and beyond", *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, 1986, pp. 188 – 195.

[BC2] Brassard, G. and Crépeau, C., "Sorting out zero-knowledge", *Advances in Cryptology: EUROCRYPT '89 Proceedings*, Springer–Verlag, to appear.

[BCY] Brassard, G., Crépeau, C. and Yung, M., "Everything in **NP** can be argued in *perfect* zero-knowledge in a *bounded* number of rounds", *Proceedings of the 16th International Colloquium on Automata, Languages and Programming*, Springer–Verlag, 1989, pp. 123 – 136.

[C] Chaum, D., "Demonstrating that a public predicate can be satisfied without revealing any information about how", *Advances in Cryptology: CRYPTO '86 Proceedings*, Springer–Verlag, 1987, pp. 195 – 199.

[FFS] Feige, U., Fiat, A. and Shamir, A., "Zero knowledge proofs of identity", *Journal of Cryptology*, Vol. 1, no. 2, 1988, pp. 77 – 94.

[F] Fortnow, L., "The complexity of perfect zero-knowledge", *Proceedings of the 19th ACM Symposium on Theory of Computing*, 1987, pp. 204 – 209.

[GMW] Goldreich, O., Micali, S. and Wigderson, A., "Proofs that yield nothing but their validity and a methodology of cryptographic protocol design", *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, 1986, pp. 174 – 187. The final paper is to appear in the *Journal of the ACM*.

[GMR] Goldwasser, S., Micali, S. and Rackoff, C., "The knowledge complexity of interactive proof systems", *SIAM Journal on Computing*, Vol. 18, no. 1, 1989, pp. 186 – 208.

[TW] Tompa, M. and Woll, H. "Random self-reducibility and zero-knowledge proofs of possession of knowledge", *Proceedings of the 28th IEEE Symposium on Foundations of Computer Science*, 1987, pp. 472 – 482.