Constant-round perfect zero-knowledge computationally convincing protocols

Gilles Brassard *

Département d'informatique et de R.O.
Université de Montréal
C.P. 6128 succ. "A"
Montréal, Québec CANADA H3C 3J7

Claude Crépeau †

Laboratoire de Recherche Informatique
Université de Paris-Sud
Bâtiment 490
91405 Orsay FRANCE

Moti Yung

IBM T. J. Watson Research Center
Yorktown Heights, NY, U.S.A.

15 March 1989

*Supported in part by Canada NSERC grant A4107.

†Supported in part by NSERC Postgraduate and Postdoctorate Scholarships. Some of this research was performed while visiting the IBM Almaden Research Center.

Abstract

A perfect zero-knowledge interactive protocol allows a prover to convince a verifier of the validity of a statement in a way that does not give the verifier any additional information [GMR, GMW]. Such protocols take place by the exchange of messages back and forth between the prover and the verifier. An important measure of efficiency for these protocols is the number of rounds in the interaction. In previously known perfect zero-knowledge protocols for statements concerning **NP**–complete problems [BCC], at least k rounds were necessary in order to prevent one party from having a probability of undetected cheating greater than 2^{-k}. In this paper, we give the first perfect zero-knowledge protocol that offers arbitrarily high security for any statement in **NP** with a constant number of rounds. The protocol is *computationally convincing* (rather than statistically convincing as would have been an interactive proof–system in the sense of Goldwasser, Micali and Rackoff) because the verifier's belief in the prover's claim is partly based on the assumption that it is feasible to find a prime p with known factorization of $p-1$ such that it is infeasible to compute discrete logarithms modulo p even for someone who knows the factors of $p-1$. Our protocol can also be based on the more general assumption that *one-way certified group action* exist, which is a new concept introduced in this paper. It is still open whether it would be sufficient to assume the existence of one-way functions in order to obtain perfect zero-knowledge computationally convincing interactive protocols for all statements in **NP**.

1 Introduction

Much excitement was caused when it was discovered in 1986 by Goldreich, Micali and Wigderson that all statements in **NP** have *computational* zero-knowledge interactive proof–systems (under the assumption that secure encryption functions exist) [GMW]. See also [BC1]. Such proof–systems, a notion formalized by Goldwasser, Micali and Rackoff a few years previously, allow an infinitely powerful (but not trusted) prover to convince a probabilistic polynomial-time verifier of the validity of a statement in a way that does not convey any *polynomial-time-usable* knowledge to the verifier, other than the validity of the statement [GMR]. Informally, this means that the verifier should not be able to generate anything in probabilistic polynomial time after having participated in the protocol, that he could not have generated by himself without ever talking to the prover (from mere belief that the statement is true). We say of such proof–systems that they are *statistically convincing* because the verifier is convinced by overwhelming statistical evidence: no matter how she [1] proceeds, the prover is almost certain to be caught cheating if she tries to convince the verifier of a false statement. A more precise definition (not needed for this paper) can be found in [GMR].

A result similar to those of [GMW, BC1] was obtained independently by Chaum, but under a very different model, which emphasizes the unconditional privacy of the prover's secret information, even if the verifier has unlimited computing power [Ch]. Independently, Brassard and Crépeau considered a model (compatible with Chaum's) in which all parties involved are assumed to have reasonable computing power, and they also obtained a protocol unconditionally secure for the prover (meaning that the prover's safety did not depend on unproved cryptographic assumptions) [BC2]. On the other hand, these protocols are *not* statistically convincing (hence they are not proof–systems in the terminology of [GMR]) because the prover could cheat without fear of detection if she had unlimited computing power or, more importantly in practice, if she could break an unproved cryptographic assumption in real-time. For this reason, we say of these protocols that they are *computationally convincing*. (Such protocols have also been termed *arguments* [BC3] in order to distinguish them from proof–systems; Goldreich has proposed calling them *computationally–sound* proofs; we follow here a terminology recently proposed by Chaum.)

The difference between these results is important because all the information on the prover's secret *is given to the verifier* in the protocols of [GMW, BC1], albeit in enciphered form. Hence, the verifier can have access to the prover's secret information, provided he can break the cryptographic assumption — perhaps by luck — or if he is willing to undergo an infeasible computation. Such attacks from the verifier can even be performed *off-line*, i.e. any time after the protocol has been completed. In contrast, no information at all (even in the sense of Shannon's information theory [S]) is given to the verifier in the protocols of [Ch, BC2], except with an exponentially small probability. This exponentially small probability of cheating for the verifier was subsequently removed by Brassard, Chaum and Crépeau [BCC], thanks to an

[1] It is convenient to give distinct and definite genders to the participants of our protocols. The prover will be referred to as a "she" (think of P as Peggy) and the verifier will be referred to as a "he" (V as Vic). Other actors to be introduced later are the simulator (Section **??**), referred to as an "it", and the originator and the receiver in bit commitment schemes (Section **??**). The case of the latter two is more complex because the originator's role is usually played by the prover but sometimes by the verifier. Out of context, the originator will be a "she" and the receiver will be a "he", but this will switch when the originator is definitely the verifier.

idea of Damgård [CDG] (and independently of Boyar, Krentel and Kurtz [BKK]), yielding in effect a *perfect* zero-knowledge protocol in the terminology of [GMW]. This implies that the prover's safety would still be guaranteed indefinitely, even if strong organizations with unknown computing power and algorithmic knowledge were to try to extract her secret. Thus, the crucial difference between computational and perfect zero-knowledge is that in the former case the verifier may obtain information on the prover's secret through the interaction, but he cannot make use of it in polynomial time (unless the cryptographic assumption fails), whereas in the latter case, the verifier obtains no information whatsoever on the prover's secret (beyond its existence and the fact that the prover knows it). To summarize, [GMW, BC1] give computational zero-knowledge statistically convincing interactive protocols, whereas [BCC] gives a perfect zero-knowledge computationally convincing interactive protocol.

What about the best of both worlds? Unfortunately, Fortnow proved that is in unlikely that there is a free lunch [F]: the prover's increased safety in [Ch, BC2, BCC] inevitably requires that an infinitely powerful prover (in fact exponentially powerful would suffice) could cheat the verifier (unless the polynomial hierarchy collapses). This is why a perfect zero-knowledge statistically convincing interactive protocol is unlikely to exist for **NP**–complete problems, even under a reasonable cryptographic assumption.

The main motivation behind the work of [GMW, BC1, Ch, BC2, BCC] was a quest for *generality*: what are the most general statements that can be handled by (perfect) zero-knowledge interactive protocols if little attention is paid to efficiency? Other researchers were willing to sacrifice generality on the altar of efficiency. The best known instance of this approach is Feige, Fiat and Shamir's identification system [FFS], which handles an *ad hoc* problem relevant to the purpose of identification, but was not designed to handle statements about **NP**–complete problems. One reason why the FFS scheme is so attractive in practice is that it requires only a few rounds of communication between the prover and the verifier. A constant-round perfect zero-knowledge statistically convincing interactive protocol was also known for the specific problem of graph *non*-isomorphism, due to Goldreich, Micali and Wigderson [GMW]. In sharp contrast, the more general protocols of [GMW, BC1, Ch, BC2, BCC] require an arbitrarily high number of rounds in order to achieve an arbitrarily high level of confidence. This paper addresses the following question: Is it possible to combine generality and arbitrarily high confidence with a constant number of rounds? Provided that one-way certified group actions exist (see Section **??**), our answer is that any **NP** statement can be handled by a three-round perfect zero-knowledge computationally convincing interactive protocol. (By one "round", we mean two "moves": one message sent by the verifier followed by one message sent by the prover.)

Similar questions have been investigated by other researchers. Goldreich and Kahn [GKa], and independently Naor and Yung [NY] have developed constant-round computational zero-knowledge statistically convincing interactive protocols for all **NP** statements. In contrast, we describe here the first constant-round *perfect* zero-knowledge interactive protocol, which is of course merely computationally convincing. Our work was first presented at the 16th International Colloquium on Automata, Languages and Programming (ICALP), held in July 1989 [BCY]. Shortly thereafter, Feige and Shamir introduced an entirely different solution to the same problem at the CRYPTO '89 conference, held in August 1989 [FS]. Most of Feige and Shamir's work deals with a computationally convincing interactive protocol that is merely computational zero-knowledge under the very weak assumption that one-way functions exist, but they also show how to make it perfect zero-knowledge under a stronger assumption essentially identical to the one we need here.

Further developments are also worth mentioning. After reading our ICALP paper [BCY], Bellare, Micali and Ostrovsky discovered a beautiful constant-round perfect zero-knowledge statistically convincing interactive protocol for graph isomorphism and, more generally, for any random self-reducible problem [BMO]. Their protocol does not depend on any unproved assumptions. They achieved this by building upon our technique, and adding several very clever ideas, in particular in the design of their simulator. Of course, because of Fortnow's result, their protocol cannot be extended to **NP**–complete problems unless the polynomial hierarchy collapses.

2 Sequential versus parallel templates

More precise definitions of what exactly is a perfect zero-knowledge computationally convincing interactive protocol are given in Section **??**, just before we describe our final protocol and prove that it is correct. In the mean time, we believe that it is preferable to keep the discussion at a more intuitive level.

The protocols of [GMW, BC1, Ch, BC2, BCC] all follow the same basic template, which we shall refer to as the *sequential* approach. Very abstractly, it is illustrated as Protocol 1 below. Here, k is a confidence parameter that must be agreed upon in advance between the prover and the verifier (perhaps during the initialization step).

Protocol 1 (the sequential approach)

1: perhaps some initialization

2: **for** $i \leftarrow 1$ **to** k **do**
    The prover commits herself by sending x_i to the verifier
    The verifier sends challenge y_i to the prover
    The prover meets the challenge by sending z_i to the verifier.

Each time round the loop, the prover places herself in a situation in which several challenges (2 in [BC1, Ch, BC2, BCC], many in [GMW]) could be issued by the verifier. This is done in a way that the prover can meet all of these challenges if and only if she is honest in her claim, but meeting any specific one of them yields worthless information to the verifier. Because the prover is required to meet only one challenge in each round, and because each round is independent from the others, the verifier learns nothing about her secret. Because the prover cannot predict ahead of time which challenge will be issued, the verifier's confidence in the prover's claim increases exponentially with the number of rounds.

In order to reduce the number of rounds to a constant, Protocol 2 comes immediately to mind. We shall refer to it as the *parallel* approach.

Protocol 2 (the parallel approach)

1: perhaps some initialization

2: The prover commits herself by sending $x_1, x_2, \ldots, x_k$ to the verifier

3: The verifier sends challenges $y_1, y_2, \ldots, y_k$ to the prover

4: The prover meets the challenges by sending $z_1, z_2, \ldots, z_k$ to the verifier.

It is easy to prove that the parallel approach cannot help the prover cheat. At first, it seems equally certain that if the verifier could not squeeze additional knowledge out of the sequential protocol, running the protocol in parallel cannot possibly help him to do so. However, nothing is certain in this world except death and taxes, and indeed *this intuition is false* (as pointed out in [GMW] in relation with their protocol for graph isomorphism). This is because the parallel approach makes it possible for the verifier to choose each challenge as a function of the entire set of prover's commitments. This may seem harmless, but there are exponentially many possible challenge vectors. If this choice of challenges is made according to a one-way hash function, it makes it possible for the verifier to obtain information (the transcript of his conversation with the prover) that he could apparently not have obtained efficiently without access to the prover (or to someone in possession of her secret information). Goldreich and Krawczyk have indeed recently proved that the direct parallelization of the [GMW] protocol for graph isomorphism is *not* perfect zero-knowledge if the verifier is allowed to be non-uniform whereas the simulator (see Section **??**) is restricted to being black-box [GKr]. Whether this could in some cases help the verifier compute the prover's secret is a most interesting open question.

3 Bit commitment schemes

Central to our parallel protocol for any **NP**-statement is the notion of *trap-door* bit commitment scheme, which was first introduced in [BCC] under the more poetic name of *chameleon blobs*. The phrase "trap-door bit commitment scheme" was coined by Shamir [FS] after reading [BCC]. We prefer it because it is more self-explanatory.

The purpose of a *bit commitment scheme* (not necessarily trap-door) is to allow one party, the *originator*, to commit to the value of a bit in a way that prevents the other party, the *receiver*, from learning it without the first party's help, but also in a way that prevents the first party from changing its value. At any time after she has committed to a bit, the originator can show the receiver which bit she had committed to, a process known as *opening* the bit commitment. Of course, she should not be able to cheat by "changing her mind", i.e. showing the wrong bit. In most zero-knowledge interactive protocols, the originator is the prover, but we shall see that this is not necessarily so.

Bit commitment schemes can be implemented in a great many ways [BCC]. When they are based on cryptography and computational complexity (rather than physical envelopes or other devices such as quantum blobs [BB2]), they are necessarily imperfect. This imperfection can happen in two very different ways. If it is impossible for the originator to change her commitments in the receiver's back, then it can be at best infeasible (i.e. within a reasonable amount of time) for the receiver to determine the bits committed to without the originator's help. Conversely, if it is impossible for the receiver to determine these bits, then it can be at best infeasible for the originator to change her mind about a bit she had committed to when (and if) she subsequently decides to open the commitment. All the known perfect zero-knowledge interactive protocols for statement about **NP**–complete problems are based on this second type of bit commitment scheme, and our new protocol is no exception.

Let us illustrate this notion by reviewing the first bit commitment scheme that allowed for a perfect zero-knowledge computationally convincing interactive protocol for all **NP** statements [BCC]. This scheme, which is central to our new constant-round protocol as well, was invented independently by Damgård [CDG] and by Boyar, Krentel and Kurtz [BKK]. In Section **??**, we shall see that this scheme is a special case of the broad class of bit commitment schemes based on one-way certified group actions.

Let us first review some elementary number theory [K]. If p is a prime number, let us denote by $\mathbf{Z}_p^*$ the multiplicative group of non-zero integers modulo p, i.e. $\{1, 2, \ldots, p-1\}$ under multiplication modulo p. Similarly, let us denote by $\mathbf{Z}_{p-1}$ the additive group of integers modulo $p-1$, i.e. $\{0, 1, \ldots, p-2\}$ under addition modulo $p-1$. Notice that $\mathbf{Z}_p^*$ and $\mathbf{Z}_{p-1}$ contain the same number of elements. For any integers a, b and c such that $a \not\equiv 0 \pmod{p}$ and $b \equiv c \pmod{p-1}$, we have by Fermat's theorem that $a^b \equiv a^c \pmod{p}$. Therefore, it makes sense to speak of x^i for $x \in \mathbf{Z}_p^*$ and $i \in \mathbf{Z}_{p-1}$. An element α of $\mathbf{Z}_p^*$ is called a *generator* of $\mathbf{Z}_p^*$ if each element of $\mathbf{Z}_p^*$ can be obtained as a power of α. Therefore, if α is a generator, the function $exp_\alpha : \mathbf{Z}_{p-1} \to \mathbf{Z}_p^*$ defined by $exp_\alpha(i) = \alpha^i$ establishes a one-one correspondence.

In order to set up the bit commitment scheme, the originator and receiver initially agree on a large prime p for which they both know the factorization of $p-1$ (finding large primes p with known factorization of $p-1$ can be done efficiently in practice [M]). They also agree on a generator α of $\mathbf{Z}_p^*$. Thanks to their knowledge of the factors of $p-1$, they can both verify with certainty that p is a prime and that α is a generator of $\mathbf{Z}_p^*$. Moreover, the density of generators is high enough that one can be found reasonably efficiently by random trial and error. Actually, the parameters p and α need not be changed each time a bit commitment scheme has to be set up. Rather, they could be in the public domain (together with the factorization of $p-1$) after having been selected once and for all by an authority that does not need to be trusted. Given any $i \in \mathbf{Z}_{p-1}$, it is easy to compute α^i efficiently by a divide-and-conquer approach, but no efficient algorithm is known to invert this process (even if the factors of $p-1$ are known, provided they are not too small [PH]), an operation known as *extracting the discrete logarithm.*

Once the parameters p and α have been agreed upon, the receiver chooses a random $s \in \mathbf{Z}_p^*$ and gives it to the originator. We assume the *certified discrete logarithm assumption*, namely that the originator is not capable of computing the discrete logarithm of s while the protocol is in progress ("certified" because the factors of $p-1$ are known to all parties in order that α be a certified generator — since this could make computing the discrete logarithm easier, this assumption is stronger than the usual discrete logarithm assumption [BM]).

In order to commit to bit $x \in \{0, 1\}$, the originator selects a random $r \in \mathbf{Z}_{p-1}$ and she computes $b = \alpha^r s^x$. She gives b to the receiver but she keeps r as her secret *witness.* For convenience, we shall refer to b as a *blob.* Subsequently, if the originator wishes to convince the receiver that b was a commitment to bit x (recall that this operation is known as *opening* the blob), she simply shows him the corresponding witness r. The receiver can then check that indeed $b = \alpha^r s^x$.

Because the function exp_α is a one-one correspondence, any element of $\mathbf{Z}_p^*$ can be used by the originator as commitment to 0 just as well as to 1, depending only on which witness she knows. Moreover, all commitments give rise to blobs that are randomly and independently distributed according to the uniform distribution over $\mathbf{Z}_p^*$. Therefore, it is information-theoretically impossible for the receiver to distinguish a commitment to 0 from a commitment to 1, regardless of his computing power. On the other hand, the originator is able to open a given blob both ways if and only if she knows (or can efficiently compute) the discrete logarithm of s, which we assumed to be infeasible for her. Therefore, it seems clear that the originator cannot cheat, as argued in [BCC], if indeed computing this discrete logarithm is infeasible for her. It came as a great surprise to us that this is not so and that this bit commitment scheme allows a cheating originator to get away with it in some situations. We shall discuss this issue later when we address the problem of so-called EPR–blobs (see Section **??**).

Using this bit commitment scheme (with the prover as originator of all commitments), a protocol for satisfiability is given in [BCC]. This protocol is reviewed in Section **??**. Assuming the certified discrete logarithm assumption, this is indeed a perfect zero-knowledge computationally convincing interactive protocol despite the weakness in the bit commitment scheme alluded to in the previous paragraph, but of course only if it is performed sequentially (as required in [BCC]). Nevertheless, even the parallel version of this protocol would be perfect zero-knowledge if only the verifier (in fact the simulator, to be technically exact — see Section **??**) had the key to its trap-door, a notion that we now explain.

A bit commitment scheme has the *trap-door property* if, in addition to the usual requirements of bit commitment schemes, there exists a secret, known as the *key* to the trap-door, that would allow the originator to cheat her commitments any time she wants if only she knew this key. More precisely, knowledge of this key would make it possible for her to offer fake "commitments" that she could subsequently "open" either way at her choice of the moment, and these fake commitments are information-theoretically indistinguishable from genuine ones. A moment's thought suffices to see that the bit commitment scheme based on the certified discrete logarithm assumption is indeed trap-door, and that its key is the discrete logarithm of

s. (You may think that we will *never* allow the originator of commitments to know this key, but in fact it is going to be just the opposite!)

In other words, the computationally convincing interactive protocol of [BCC] remains perfect zero-knowledge even if run in parallel, provided that the verifier knows the discrete logarithm of s. But this requirement is apparently easy to fulfill because the verifier in the global protocol is the receiver of the bit commitment scheme, hence it is his prerogative to choose the parameter s. Instead of choosing it randomly in $\mathbf{Z}_p^*$, he could choose j at random in $\mathbf{Z}_{p-1}$ and compute $s = \alpha^j$.

So are we done? Not quite because the prover would have no reason to trust the verifier to choose s in a way that he (the verifier) would know the key to the bit commitment scheme trap-door. The obvious way to solve this difficulty would be to require that the verifier convince the prover that he knows the discrete logarithm of s in the initialization of Protocol 2. Naturally, this initialization would have to be achieved with a perfect zero-knowledge protocol (in which prover and verifier switch roles temporarily) since the discrete logarithm of s must remain hidden from the prover. Unfortunately, the protocols discovered several years ago for this task inherently require an unbounded number of rounds [CEG, CEGP]! As mentioned at the end of Section **??**, inspired by a previous version of the current paper [BCY], Bellare, Micali and Ostrovsky have recently shown how to achieve constant-round perfect zero-knowledge statistically convincing interactive protocols for any random self-reducible problem, including the discrete logarithm [BMO]. Using their protocol as the initialization of Protocol 2 would indeed solve our problem. Nevertheless, it would result in a more complicated protocol, requiring more rounds. In particular, the simulator (see Sections **??** and **??**) would be significantly harder to work out. Let us now proceed to describe our original protocol, without using subsequent ideas that were derived from it.

Our final solution for a constant-round protocol is to use a trap-door bit commitment scheme such that the verifier cannot cheat if he knows the key for one reason, and such that he cannot cheat either if he does *not* know the key, but for an entirely different reason. Let us now sketch this idea intuitively before describing the final protocol formally.

4 Forcing the verifier to commit

From now on, we assume that the trap-door bit commitment scheme based on the certified discrete logarithm assumption is used, as described in the previous section. All protocols start with the prover and verifier agreeing once and for all on a suitably large prime p with known factorization of $p-1$, with a generator α of $\mathbf{Z}_p^*$, and with the choice of some $s \in \mathbf{Z}_p^*$ by the verifier. However, we do *not* restrict the verifier in his way of choosing s. He can choose it randomly in $\mathbf{Z}_p^*$, he can choose its discrete logarithm at random and compute s from it, or he can follow any other (efficient) strategy.

Once all the bit commitment scheme parameters are agreed upon, either party can act as originator for commitments (recall that the prover used to be the only originator). Notice that we have slightly departed from the formal description of the discrete logarithm bit commitment scheme since we had previously said that the receiver had to choose s, whereas now it is chosen by the verifier who can also take the role of the originator. Of course, such commitments are bogus if the originator is the verifier and if he has chosen s in a way that he knows its discrete logarithm. Allowing such bogus commitments may sound crazy, but in fact this is perhaps the most crucial idea in this paper.

Why is it interesting for the verifier to originate bit commitments? As we have already discussed, the reason why Protocol 2 is not perfect zero-knowledge is that the verifier can choose his challenges $y_1, y_2, \ldots, y_k$ as some complex function of $x_1, x_2, \ldots, x_k$. This is no longer possible if the verifier is forced to select his challenges *before* seeing the x_i's. But of course, the prover must not be allowed to know what the challenges will be when he decides which x_i's to offer to the verifier. Hence, a solution is to require the verifier to commit to his challenges before what was step 2 in Protocol 2, and to have him open his commitments afterwards. The intent is to force the challenges to be independent from the x_i's but still have the x_i's independent from the challenges. This leads to Protocol 3 below, which is *almost* the final solution.

Protocol 3 **(forcing the verifier to commit)**

1: The prover and the verifier agree on p, α and k

2: The verifier chooses s in $\mathbf{Z}_p^*$; he sends s to the prover
The verifier also chooses bits $y_1, y_2, \ldots, y_k$ and commits to them

3: The prover commits herself by sending $x_1, x_2, \ldots, x_k$ to the verifier

4: The verifier opens challenges $y_1, y_2, \ldots, y_k$ for the prover

5: The prover meets the challenges by sending $z_1, z_2, \ldots, z_k$ to the verifier.

Now, the verifier is faced with a tough choice if he wants to squeeze information from the prover: either he chooses s in step 2 such that he knows its discrete logarithm, or he does not (in fact, there are other more exotic strategies for the verifier, and they will be taken into account later). If he does, his commitments to $y_1, y_2, \ldots, y_k$ are bogus, which seems to make Protocol 3 no better than Protocol 2 since the verifier can wait to see the x_i's before having really to decide on his challenges. But that's fine because Protocol 2 *is* perfect zero-knowledge in the case in which the verifier knows the key to the bit commitment scheme trap-door. Conversely, if the verifier chooses s in a way that he does *not* know its discrete logarithm, then his commitments to $y_1, y_2, \ldots, y_k$ are genuine, so that they are independent from the x_i's and the protocol is perfect zero-knowledge as well. In other words, the full title of this section should have been "Forcing the verifier to commit… either to his challenges or to the fact that he knows the key to the trap-door". Either way we win! And indeed Protocol 3 is perfect zero-knowledge, as the enthusiastic reader may wish to prove.

Unfortunately, Protocol 3 *still* does not work for a rather subtle reason: it is no longer a computationally convincing interactive protocol because the prover can cheat without ever needing to find the discrete logarithm of s! Although it is true that the prover is not capable of opening any of her blobs in more than one way (assuming she does not know and cannot compute the discrete logarithm of s), it *is* possible for her to create blobs that she cannot open *at all* at the time of their creation, but that she will be able to open later. When the verifier opens his own commitments at step 4, the prover will be able to open her blobs to show bits related to those that the verifier had committed to. As a simple example, assume that the verifier commits to some challenge y in step 2 by choosing a random $r \in \mathbf{Z}_{p-1}$ and computing $e = \alpha^r s^y$. After the verifier has given his commitment e to the prover, the prover can create a blob x by computing $x = \alpha^q e$ for a randomly chosen $q \in \mathbf{Z}_{p-1}$. Clearly, the prover cannot open this blob either way yet. Nevertheless, the verifier will reveal r to the prover in step 4, and this will enable the prover to "open" her blob x as bit y by showing "witness" $q + r$ since $x = \alpha^{q+r} s^y$ (do not forget that q and r are members of $\mathbf{Z}_{p-1}$, hence the addition is done modulo $p - 1$). This ability of the prover to relate the value of some of her blobs to the challenges she will subsequently be issued spells doom on the protocol of [BCC]. Actually, this is not a problem with Protocol 3 itself (an observation due to Shafi Goldwasser). Rather, it points to the imperfection we had promised earlier about the discrete logarithm bit commitment scheme. It is *not enough* for a bit commitment scheme to forbid the originator from opening a blob both ways. It must also be that the originator knows already which (unique) way she can open a blob *at the moment of its creation*. In other words, the originator should not be allowed to "commit" to a bit that she does not yet have in mind.

In order to solve this ultimate difficulty and obtain our final constant-round perfect zero-knowledge computationally convincing interactive protocol, we must force the prover to emit only blobs that she knows how to open at the time of their creation. Let x be a blob created by the prover. We say that it is a 0–blob (resp. a 1–blob) if the prover can exhibit a z such that $x = \alpha^z$ (resp. $x = \alpha^z s$). As argued previously, no prover-created blob can be simultaneously a 0–blob and a 1–blob unless the prover knows (or can compute easily) the discrete logarithm of s. However, a blob can be neither a 0–blob nor a 1–blob if it was not created "according to the rules" by the prover. We call such blobs "EPR–blobs" by analogy with the way the Einstein–Podolsky–Rosen "paradox" of quantum physics [AGR] allows cheating the quantum coin-tossing protocol [BB1]. Again, the danger of EPR–blobs is that they could subsequently become either 0–blobs or 1–blobs when the prover obtains further information from the verifier (i.e. when the verifier opens his commitments to his challenges).

The following *blob certification subprotocol* is inspired by Benaloh's cryptographic capsules [Be], and a similar idea was used in the perfect zero-knowledge statistically convincing interactive protocol for graph *non*-isomorphism due to Goldreich, Micali and Wigderson [GMW]. It allows the prover to convince the verifier that a given blob is not EPR, but does not reveal anything as to whether it is a 0–blob or a 1–blob. Let x be a non-EPR–blob, which we shall call the *actual* blob. Let $c \in \{0,1\}$ be such that x is a c–blob. Let z be the prover's witness that would allow her to open this blob, i.e. $x = \alpha^z s^c$. In order to convince the verifier that she can open blob x, the prover creates k additional *control* blobs $u_1, u_2, \ldots, u_k$ by randomly selecting k bits $b_1, b_2, \ldots, b_k$ together with $v_1, v_2, \ldots, v_k$ in $\mathbf{Z}_{p-1}$, and computing $u_i = \alpha^{v_i} s^{b_i}$, $1 \le i \le k$. She gives all these control blobs to the verifier. At this point, the verifier selects k random challenges $h_1, h_2, \ldots, h_k \in \{0,1\}$ and sends them to the prover. For each $h_i = 0$, the prover opens control blob u_i by showing b_i and v_i. For each $h_i = 1$, the prover shows that she can open the actual blob x if and only if she can open control blob u_i. This is done as follows:

- if $c = b_i$, the prover computes $w_i = v_i - z$ and gives it to the verifier, who checks that $u_i = \alpha^{w_i} x$;
- if $c \ne b_i$, the prover computes $w_i = v_i + z$ and gives it to the verifier, who checks that $u_i x = \alpha^{w_i} s$.

If in fact x is an EPR–blob, the only way the prover could fool the verifier in this blob certification subprotocol would be to guess exactly the challenges $h_1, h_2, \ldots, h_k$ to come. This would allow her to prepare control blob u_i "honestly" when $h_i = 0$ but otherwise to compute $u_i = \alpha^{w_i} x$ if $b_i = 0$ or $u_i = \alpha^{w_i} s x^{-1}$ if $b_i = 1$ for a randomly chosen $w_i \in \mathbf{Z}_{p-1}$. If any of the h_i's had been guessed incorrectly, the prover would either be caught unable to open control blob u_i (if in fact $h_i = 0$) or unable to relate x to u_i (if $h_i = 1$). The probability of such successful guessing is 2^{-k}. On the other hand, it is clear that the verifier learns nothing about c by learning either the value of b_i when $h_i = 0$ or whether or not $c = b_i$ when $h_i = 1$.

You may have noticed that this blob certification subprotocol is run in parallel: the prover gives all the control blobs to the verifier, and only then does the verifier give his string of challenges to the prover. This is precisely where the infernal circle is broken. Even better, any number of instances of the blob certification subprotocol can take place in parallel. Consider ℓ actual blobs for which the prover wishes to convince the verifier that none of them is EPR. For this, she can provide k control blobs for each actual blob before requesting the whole batch of $\ell \times k$ challenges from the verifier. A formal proof that the verifier learns nothing about the prover's secret even when this process is run in parallel will be implicit when we prove in Section **??** that our final protocol (Section **??**), which uses the blob certification subprotocol, is perfect zero-knowledge.

The final protocol is exactly like Protocol 3, except that the blob certification subprotocol is used between steps 3 and 4 for each and every blob that the prover had issued at step 3. In this way, it becomes impossible (except with probability 2^{-k}) that the information provided by the verifier at step 4 might help the prover cheat. We shall now proceed to describe the final protocol in detail, instead of the high-level abstraction used so far in Protocols 1–3. Before this can be done, however, a review of the *basic BCC–protocol* is necessary [BCC]. Moreover, for the sake of self-containment, we also need to review the formal definitions of perfect zero-knowledge and of computationally convincing interactive protocols.

5 Notation and definitions

5.1 Interactive protocols

The notion of interactive protocol was originally formalized by Goldwasser, Micali and Rackoff [GMR]. In this section, we review the definitions and modify them when necessary for our purpose. An *interactive protocol* takes place by the exchange of messages back and forth between two parties, known as the *prover* and the *verifier*. The purpose of the protocol is for the prover to convince the verifier of the validity of a statement such as "Boolean expression Ψ is satisfiable" or "graphs G and H are not isomorphic", or "I know the factorization of large integer n".

The protocol's specification prescribes the behaviour of the "honest" parties, but sometimes we must deal also with parties that deviate arbitrarily from their prescribed behaviour. Such dishonest parties are called *cheaters*. It is convenient to think of the prover and verifier (honest or cheating) as probabilistic algorithms. These algorithms take turns at being active. One of them is active initially. Whenever it wishes, the active party can terminate the entire protocol, or activate the other party before going to sleep. When the latter situation occurs, the activating party usually passes a message to the activated party. Normally, it is the verifier that decides to terminate the protocol, in which case he also chooses whether to *accept* or *reject*.

In the original definition of interactive protocols by Goldwasser, Micali and Rackoff [GMR], the verifier (honest or cheating) is restricted to being probabilistic polynomial time. In the context of *perfect* zero-knowledge (Section **??**), which concerns us in this paper, we believe that this is unnecessarily restrictive. Hence, we impose no computational restrictions on cheating verifiers. Nevertheless, we ask that the honest verifier be efficient since otherwise the protocol would be of no practical value. On the other hand, the prover of [GMR] is assumed to have unlimited computing power. We cannot accommodate such powerful provers in the context of computationally convincing interactive protocols (see below). However, we do not require specifically that the prover be probabilistic polynomial time because this would be an asymptotic restriction, which may be totally irrelevant in real-life instances. Rather, we make the assumption that there is a specific instance of a computational task that the prover cannot perform while the protocol is in progress. More details are given below.

To any given run of an interactive protocol, there corresponds a *confidence parameter* k, agreed upon between the prover and the verifier. It must be the case that the expected time to carry out the protocol between the honest parties grows at worst linearly with the value of k. The interactive protocol is *statistically complete* if, whenever the prover is honest, the probability that the honest verifier will terminate and accept is at least $1 - 2^{-k}$. It is *perfectly* (or *unconditionally*) complete if this probability is exactly 1, as is the case for most interactive protocols in the literature, including our own given in Section **??**. Notice that we do not require that a dishonest verifier must accept with high probability when the prover is honest, because of course a very stubborn cheating verifier could reject no matter what happens.

Let us now consider an arbitrary dishonest prover, who is trying to convince the verifier of a false statement. Of course, such a cheating prover is not required to follow her prescribed share of the protocol; she can do whatever she wants in her attempts to fool the verifier. Let us nevertheless assume the existence of a specific computational task that the prover cannot perform, except with negligible probability $\wp$, during the time necessary to carry out the interactive protocol. The protocol is *computationally convincing* (with respect to that assumption) if the probability that the honest verifier, when interacting with this cheating prover, will accept the prover's claim is at most $\wp + 2^{-k}$. In other words, the only way the cheating prover can succeed is either by being exponentially lucky (with probability 2^{-k}), or by successfully carrying out (with probability $\wp$) the computational task that we assumed her incapable of doing.

It should be emphasized that the computational limitation imposed on the prover is *concrete*, rather than being asymptotic as if we had simply requested her to be a probabilistic polynomial-time algorithm. Even if $\mathcal{P} = \mathcal{NP}$, it may still be infeasible for the prover to extract a large enough discrete logarithm while the interactive protocol is in progress (except with negligible probability).

Let us finally remark that most published work that deal with interactive protocols do not have this confidence parameter k. Rather, they insist that the probability of successful cheating should be bounded by a function of the size of the prover's claim. We feel that this is inadequate because there is no direct link between the importance for the verifier of not being fooled and the size of the statement under consideration (except perhaps that if the statement is short enough, the verifier might be able to handle it without help from the prover). Moreover, using k larger than, say, 300 is a ridiculous waste of time, whereas instances of size larger than 300 could be reasonable in practice.

5.2 Perfect zero-knowledge

The notion of (computational) *zero-knowledge* was also introduced by Goldwasser, Micali and Rackoff [GMR]. In this paper, we only deal with the simpler and stronger notion of *perfect* zero-knowledge, which was introduced by Goldreich, Micali and Wigderson [GMW].

Consider an interactive protocol between the prover and the verifier. Assume that the prover is honest, but let us consider a cheating verifier that can deviate arbitrarily from his prescribed behaviour. (In the context of [GMW], the verifier is restricted to being probabilistic polynomial time; we do not impose any such restrictions here.) Intuitively, the protocol is perfect zero-knowledge if, despite his potentially nasty behaviour, the verifier does not learn anything about the prover's secret, except of course that the prover's claim is valid.

In order to formalize this notion, we must introduce the concepts of the verifier's *view* and of the *simulator* [GMR]. The view of the verifier, when interacting with the honest prover, is defined as what he gets to see while the protocol is in progress, namely his own random coin flips and the messages sent to him by the prover. Of course, this view is a random variable, which depends on the verifier's and the prover's coin flips. The honest prover together with an arbitrary verifier define a probability distribution on the verifier's possible views.

We say that this view can be simulated if there is another probabilistic algorithm, known as the *simulator*, that can output views with exactly the same probability distribution without ever talking to the prover, under the sole assumption that the prover is honest. Contrary to Goldwasser, Micali and Rackoff, we insist that this simulator should *not* depend on the verifier under consideration, but we allow it access to and complete control over the verifier. By "complete control", we mean that the simulator can run the verifier, making him believe that he is interacting with the prover. At any time during this interaction, the simulator can take snapshots of the verifier or restore him to a previous state. Moreover, the simulator is given control over the coin flips of the verifier.

It is clear that the verifier does not learn anything from the prover that the simulator (who never even communicates with the prover) could not tell him. Therefore, if the simulator is efficient, the prover does not disclose anything to the verifier that he could not have computed by himself. But what do we mean by an "efficient simulator" considering that we had not restricted the verifier to being efficient? The simulator is *efficient* if the expected time it requires to output its simulated view is at most a (small) constant factor of the expected time that the real protocol between the honest prover and the verifier under consideration would have taken. In other words, the simulator is allowed to restore the verifier and run it a few times with different partial views, but the expected number of such reruns should be small.

Finally, we say that an interactive protocol is perfect zero-knowledge if such an efficient simulator exists.

5.3 An abstract description of the basic BCC–protocol

Our constant-round perfect zero-knowledge computationally convincing interactive protocol for satisfiability is inspired from a specific protocol introduced in [BCC]. Describing this *basic BCC–protocol* in detail would take up too much space, but an abstract description of how it proceeds is sufficient to understand the new techniques and ideas. We suggest that you read this subsection even if you are familiar with [BCC], because it introduces new notation.

When we write $f : X \to Y$, it means not only that f is a function from X to Y, but that this function can be computed efficiently. As an abuse of notation, f will also serve to denote an efficient algorithm for this computation. When we write $f : X \xrightarrow{\mathrm{R}} Y$, it means that f is an efficient probabilistic algorithm that associates to each $x \in X$ a probability distribution $f(x)$ over Y. When no confusion can arise, we also use $f(x)$ to denote the probabilistic result of applying algorithm f to input x.

Let Σ denote $\{0,1\}$ and Γ denote $\{0,1,\Box\}$. Given any set X, the set of finite sequences of elements of X is denoted as always by $X^\star$. (The symbol $*$ used in $\mathbf{Z}_p^*$ should not be confused with the symbol $\star$ used in $X^\star$.) If $\vec{x} \in X^\star$, we denote its length by $|x|$ and its i^{th} element by $x[i]$. For $\vec{c} \in \Sigma^\star$ and $\vec{d} \in \Gamma^\star$, we say that $\vec{c}$ and $\vec{d}$ are *compatible*, denoted $\vec{c} \asymp \vec{d}$, if $|c|=|d|$ and if, for all i, $1 \le i \le |c|$, $c[i] = d[i]$ whenever $d[i] \neq \Box$. You should think of $\Box$ as representing an unopened bit commitment. We also define the function $\Diamond : \Gamma \to \Sigma$ by $\Diamond(0) = 0$, $\Diamond(1) = 1$, and $\Diamond(\Box) = 0$. This function is extended in the natural way to $\Diamond : \Gamma^\star \to \Sigma^\star$.

For any integer n, let $\mathcal{F}_n$ denote the set of Boolean expressions on n variables and let $\mathcal{A}_n$ denote $\{\mathsf{true}, \mathsf{false}\}^n$, the set of truth assignments to n variables. Let $\mathcal{F}$ and $\mathcal{A}$ denote the infinite union of the $\mathcal{F}_n$'s and $\mathcal{A}_n$'s, respectively. To each $\Psi \in \mathcal{F}$ is associated in [BCC] a finite set $\mathcal{C}_\Psi \subseteq \Sigma^\star$ of "scrambled circuits" (it is not important to understand the "meaning" of the elements of $\mathcal{C}_\Psi$, but if you *are* familiar with [BCC], think of them as the bits appearing in a "Figure 3" [BCC, page 162] in some predetermined order). Similarly, to each $\Psi \in \mathcal{F}$ corresponds a finite set $\mathcal{D}_\Psi \subseteq \Gamma^\star$ of "partially opened circuits" (again, if you are familiar with [BCC], think of the bits of a "Figure 4" in some predetermined order, with unopened blobs represented by $\Box$). Given any $\Psi \in \mathcal{F}$ and $\vec{c} \in \Sigma^\star$, it is efficiently decidable whether or not $\vec{c} \in \mathcal{C}_\Psi$. Similarly, given any $\Psi \in \mathcal{F}$ and $\vec{d} \in \Gamma^\star$, it is efficiently decidable whether or not $\vec{d} \in \mathcal{D}_\Psi$. Moreover, for any given $\Psi \in \mathcal{F}$, all the elements of $\mathcal{C}_\Psi$ and $\mathcal{D}_\Psi$ are of the same length, which is denoted by $\ell(\Psi)$.

The key property that makes the basic BCC–protocol work is that, for any $\Psi \in \mathcal{F}$, there exist $\vec{c} \in \mathcal{C}_\Psi$ and $\vec{d} \in \mathcal{D}_\Psi$ such that $\vec{c} \asymp \vec{d}$ if and only if Ψ is satisfiable. More precisely, there are four efficient algorithms:

1. $\sigma : \mathcal{F} \xrightarrow{\mathrm{R}} \mathcal{C}$
2. $\rho : \mathcal{F} \xrightarrow{\mathrm{R}} \mathcal{D}$
3. $\gamma : \mathcal{F} \times \mathcal{C} \times \mathcal{A} \to \mathcal{D} \cup \{\perp\}$ and
4. $\delta : \mathcal{F} \times \mathcal{C} \times \mathcal{D} \to \mathcal{A} \cup \{\perp\}$

such that

1. For all $\Psi \in \mathcal{F}$, $\sigma(\Psi)$ is uniformly distributed over $\mathcal{C}_\Psi$
2. For all $\Psi \in \mathcal{F}$, $\rho(\Psi)$ is uniformly distributed over $\mathcal{D}_\Psi$
3. If $\vec{a} \in \mathcal{A}_n$ is a satisfying assignment for some $\Psi \in \mathcal{F}_n$ and if $\vec{c} \in \mathcal{C}_\Psi$, then $\gamma(\Psi, \vec{c}, \vec{a})$ is an element of $\mathcal{D}_\Psi$ compatible with $\vec{c}$. (Otherwise, $\gamma(\Psi, \vec{c}, \vec{a}) = \perp$.) Moreover, for any satisfiable $\Psi \in \mathcal{F}$ and any satisfying assignment $\vec{a}$ for Ψ, $\gamma(\Psi, \sigma(\Psi), \vec{a})$ is uniformly distributed over $\mathcal{D}_\Psi$, and
4. If $\Psi \in \mathcal{F}$, $\vec{c} \in \mathcal{C}_\Psi$, $\vec{d} \in \mathcal{D}_\Psi$ and if $\vec{c} \asymp \vec{d}$, then $\delta(\Psi, \vec{c}, \vec{d}\,)$ is a satisfying assignment for Ψ. Otherwise, $\delta(\Psi, \vec{c}, \vec{d}\,) = \perp$.

Assume that the prover wishes to convince the verifier that she knows a satisfying assignment for some $\Psi \in \mathcal{F}$. The basic BCC–protocol consists of k independent rounds taken one after the other. Round i goes as follows: The prover computes $\vec{c_i} = \sigma(\Psi)$ and she commits to each bit $c_i[j]$ by offering a corresponding blob $x_i[j]$ to the verifier. Then, the verifier issues a challenge $y_i \in \{0, 1\}$. If the challenge is "0", the prover opens $x_i[j]$ for each j, in effect showing $\vec{c_i}$ in the clear; the verifier checks that $\vec{c_i} \in \mathcal{C}_\Psi$. If the challenge is "1", the prover uses her secret satisfying assignment $\vec{a}$ to compute $\vec{d_i} = \gamma(\Psi, \vec{c_i}, \vec{a})$ and she opens only the $x_i[j]$'s corresponding to $d_i[j] \neq \Box$ (notice that in this case $d_i[j] = c_i[j]$ because $\vec{c_i} \asymp \vec{d_i}$); the verifier checks that $\vec{d_i} \in \mathcal{D}_\Psi$. In either case, if the verifier's check fails, he stops the entire protocol and rejects. If none of the k checks fail, the verifier stops and accepts.

This protocol is computationally convincing because the prover does not know in advance which challenge to expect in each round. Let $\vec{c_i}$ and $\vec{d_i}$ be the answers it would give to each challenge in round i. Assuming that the prover cannot cheat the blobs, $\vec{c_i}$ and $\vec{d_i}$ must be compatible. If $\vec{c_i} \in \mathcal{C}_\Psi$ and $\vec{d_i} \in \mathcal{D}_\Psi$, then $\delta(\Psi, \vec{c_i}, \vec{d_i})$ is a satisfying assignment for Ψ, hence Ψ is satisfiable and furthermore the prover knows or can efficiently compute a satisfying assignment for it. On the other hand, if $\vec{c_i} \notin \mathcal{C}_\Psi$ or $\vec{d_i} \notin \mathcal{D}_\Psi$, there is at least a 50% probability that the verifier will catch the prover cheating in this round. The probability of undetected cheating is thus 2^{-k} for the entire protocol, still assuming that the prover cannot cheat the bit commitment scheme.

On the other hand, this protocol is perfect zero-knowledge provided that the commitments to 0 are information-theoretically indistinguishable from commitments to 1 (which they are if we use the bit commitment scheme described earlier). In order to simulate one round of the protocol, the simulator flips a coin. If it comes up heads, the simulator generates $\vec{c} = \sigma(\Psi)$, commits to each bit in $\vec{c}$, and asks the verifier for his challenge. If the challenge is 0, the simulator opens all the blobs; otherwise, it forgets about this unlucky event and tries again. If the coin comes up tails, the simulator generates $\vec{d} = \rho(\Psi)$, sets $\vec{c} = \Diamond(\vec{d}\,)$, commits to each bit in $\vec{c}$, and asks the verifier for his challenge. If the challenge is 1, the simulator opens the blobs corresponding to $d[j] \neq \Box$; otherwise, it forgets about this unlucky event and tries again. Because commitments to 0 and commitments to 1 are indistinguishable, the verifier's challenge is independent from the simulator's coin flip. Hence, the expected number of trials for the successful simulation of one round is 2. Therefore, the simulation is perfect and all k rounds can be simulated one after the other in about twice the time it would take to carry out the real protocol.

6 The final protocol

We are now ready to provide a formal description of the final constant-round perfect zero-knowledge computationally convincing interactive protocol for satisfiability. Because satisfiability is **NP**–complete, it follows immediately that such protocols are possible for all statements in **NP**.

Recall that Σ denotes $\{0,1\}$ and Γ denotes $\{0,1,\Box\}$. Let us denote by $\beta(c)$ the pair (x,z) corresponding to a commitment to bit c, where x is the blob and z is the witness. More specifically, if the bit commitment scheme described in Section 3 is used with parameters p, α and s, then $\beta : \Sigma \xrightarrow{\mathrm{R}} \mathbf{Z}_p^* \times \mathbf{Z}_{p-1}$ and $\beta(c)$ is obtained by choosing z uniformly at random within $\mathbf{Z}_{p-1}$ and computing $x = \alpha^z s^c$. This notation is extended in the natural way to $\beta(\vec{c}\,)$ when $\vec{c} \in \Sigma^\star$. We also extend it to $\beta(\vec{d}\,)$ when $\vec{d} \in \Gamma^\star$, in which case $\beta(\vec{d}\,) = \beta(\Diamond(\vec{d}\,))$, meaning that $d[i] = \Box$ is treated (arbitrarily) as if it were $d[i] = 0$.

Similarly, let $\varphi : \mathbf{Z}_p^* \times \mathbf{Z}_{p-1} \times \Gamma \to \{\mathsf{true}, \mathsf{false}\}$ be defined such that $\varphi(x,z,c) =$ true if and only if either $c = \Box$, or $x = \alpha^z s^c$, i.e. if and only if blob x is to remain unopened, or else z is a correct witness to the fact that x is a c–blob. This notation is also extended in the natural way to $\varphi : (\mathbf{Z}_p^*)^\ell \times (\mathbf{Z}_{p-1})^\ell \times \Gamma^\ell \to \{\mathsf{true}, \mathsf{false}\}$ for any integer ℓ, in which case $\varphi(\vec{x}, \vec{z}, \vec{c}\,) = \mathsf{true}$ if and only if $x[i] = \alpha^{z[i]} s^{c[i]}$ for all i such that $c[i] \neq \Box$, $1 \leq i \leq \ell$.

Let Ψ be a satisfiable Boolean expression. Assume the prover knows a satisfying assignment $\vec{a}$ for Ψ. The following protocol is carried out. To avoid cluttering the protocol with trivial details, some obvious but necessary tests are not explicitly shown. For instance, the prover should make sure at step P1 that $s \in \mathbf{Z}_p^*$, because otherwise the verifier would immediately obtain her secret if the prover were stupid enough to proceed with the protocol when $s = 0$.

Protocol 4 (the final constant-round protocol)

: (Initialization.) The prover and verifier agree on a large prime p such that the factorization of $p-1$ is known, a generator α of $\mathbf{Z}_p^*$, and a confidence parameter k. This step does not count towards the number of rounds because it can be done ahead of time and because p and α could even be in the public domain (chosen once and for all by an authority that needs not be trusted).

V1: (The verifier sets up the bit commitment scheme parameters and "commits" to his challenges.) The verifier selects $s \in \mathbf{Z}_p^*$ and $\vec{y} \in \Sigma^k$ at random. He computes $(\vec{e}, \vec{r}\,) = \beta(\vec{y}\,)$. He sends s and $\vec{e}$ to the prover.

P1: (The prover commits to her scrambled circuits and sends control blobs with each actual blob.) For each i, $1 \leq i \leq k$, the prover computes $\vec{c_i} = \sigma(\Psi)$ and $(\vec{x_i}, \vec{z_i}) = \beta(\vec{c_i})$. For each blob $x_i[j]$, the prover chooses $\vec{b_{ij}} \in \{0,1\}^k$ at random and computes $(\vec{u_{ij}}, \vec{v_{ij}}) = \beta(\vec{b_{ij}})$. She sends all the $\vec{x_i}$'s and $\vec{u_{ij}}$'s to the verifier. (Think of $x_i[j]$ as an *actual blob* for the j^{th} bit of the i^{th} scrambled circuit, and of $\vec{u_{ij}}$ as the collection of *control blobs* for $x_i[j]$.)

V2: (The verifier challenges the actual blobs to make sure they are not EPR.) For each i and j, the verifier chooses $\vec{h_{ij}} \in \{0,1\}^k$ at random. He sends the $\vec{h_{ij}}$'s to the prover.

P2: (The prover shows that her actual blobs are not EPR.) For each i, j and m such that $h_{ij}[m] = 0$, the prover sets $w_{ij}[m] = v_{ij}[m]$ and $t_{ij}[m] = b_{ij}[m]$. For each i, j and m such that $h_{ij}[m] = 1$, the prover computes $w_{ij}[m]$ as $v_{ij}[m] - z_i[j]$ and sets $t_{ij}[m] = 0$ if $c_i[j] = b_{ij}[m]$ or she computes $w_{ij}[m]$ as $v_{ij}[m] + z_i[j]$ and sets $t_{ij}[m] = 1$ if $c_i[j] \neq b_{ij}[m]$. The prover sends all the $\vec{w_{ij}}$'s and $\vec{t_{ij}}$'s to the verifier.

... **continued on next page** ...

Protocol 4 (continued)

V3: (The verifier checks that the actual blobs were not EPR, in which case he opens his challenges.) For each i, j and m such that $h_{ij}[m] = 0$, the verifier checks that $\varphi(u_{ij}[m], w_{ij}[m], t_{ij}[m])$. For each i, j and m such that $h_{ij}[m] = 1$, the verifier checks that $u_{ij}[m] = \alpha^{w_{ij}[m]} x_i[j]$ if $t_{ij}[m] = 0$ or $u_{ij}[m] x_i[j] = \alpha^{w_{ij}[m]} s$ if $t_{ij}[m] = 1$. If any of these checks fail, the verifier stops and rejects; otherwise, the verifier sends $\vec{r}$ and $\vec{y}$ to the prover.

P3: (The prover checks that the challenges have been opened honestly and, if so, meets them.) The prover checks that $\varphi(\vec{e}, \vec{r}, \vec{y})$. If this fails, the prover quits. Otherwise, for each i such that $y[i] = 0$, the prover sends the verifier $\vec{z_i}$ and $\vec{c_i}$. For each i such that $y[i] = 1$, the prover computes $\vec{d_i} = \gamma(\Psi, \vec{c_i}, \vec{a})$, she *resets* $z_i[j]$ to 0 when $d_i[j] = \square$, and she sends $\vec{z_i}$ and $\vec{d_i}$ to the verifier.

V4: (The verifier checks that all is well.) For each i such that $y[i] = 0$, the verifier checks that $\vec{c_i} \in \mathcal{C}_\Psi$ and that $\varphi(\vec{x_i}, \vec{z_i}, \vec{c_i})$. For each i such that $y[i] = 1$, the verifier checks that $\vec{d_i} \in \mathcal{D}_\Psi$ and that $\varphi(\vec{x_i}, \vec{z_i}, \vec{d_i})$. If any of these checks fail, the verifier stops and rejects; otherwise, he stops and accepts.

7 The protocol is computationally convincing

It is obvious that Protocol 4 is perfectly complete. Let us now argue that it is computationally convincing. Assume that the prover attempts running her share of Protocol 4 on a Boolean formula Ψ that is not satisfiable. Assume furthermore that she is not capable, while the protocol is in progress, to figure out the discrete logarithm of s, as given to her by the verifier at step V1. In this case, we claim that the probability that her cheating will go undetected is at best 2^{-k} if she interacts with the honest verifier prescribed by Protocol 4.

In order to see this, notice first that her behaviour in step P1 cannot depend on the verifier's challenge vector $\vec{y}$ since the vector of commitments $\vec{e}$ is information-theoretically uncorrelated to the challenges.

In step P1, the prover can either output at least one EPR–blob $x_i[j]$, or all of the $\vec{x_i}$'s are composed only of non-EPR–blobs. In the first case, as argued in Section **??**, the probability that this will not be detected by the verifier at step V3 is 2^{-k} at best. In the second case, there can exist at most one challenge vector $\vec{y}$ for which the prover can satisfy the verifier at step V4. This is because if there is an i such that the prover can satisfy the verifier both if $y_i = 0$ (by giving $\vec{c_i}$) and if $y_i = 1$ (by giving $\vec{d_i}$), then $\vec{c_i}$ and $\vec{d_i}$ would have to be compatible by assumption that the prover cannot change her commitments and because they are not EPR. But then $\delta(\Psi, \vec{c}, \vec{d}\,)$ is a satisfying assignment for Ψ, which contradicts the assumption that Ψ is not satisfiable. Therefore, the probability that the challenge vector opened by the verifier at step V3 is precisely the one that the prover was able to cope with, is at most 2^{-k} as well.

Actually, a stronger result can be stated: even if Ψ is satisfiable, the prover will be caught cheating with probability at least $1 - 2^{-k}$ unless she actually knows a satisfying assignment for Ψ (or can compute one while the protocol is in progress), or unless she can extract the discrete logarithm of s. Therefore, this protocol constitutes a computationally convincing *proof of knowledge*. This issue will be discussed at greater length in a subsequent paper.

8 The protocol is perfect zero-knowledge

The main theorem of this paper is that the constant-round protocol of Section **??** is perfect zero-knowledge. In order to establish this, we must exhibit a simulator, and prove that it is efficient and that it produces views with exactly the correct distribution. First of all, let us see the simulator.

8.1 The simulator

In order to simulate perfectly and efficiently the conversation that the real prover would have with any verifier, we must take account of a wide range of possible behaviours for the verifier. For instance, the simulator must succeed when

1. the verifier chooses s at step V1 in a way that he knows its discrete logarithm, which allows him to give bogus commitments also at step V1 and thus postpone his choice of challenges until step V3; or

2. the verifier chooses s at step V1 such that it does not know its discrete logarithm; or

3. more subtlely, when the verifier chooses s in a way that it does not yet know its discrete logarithm at step V1, but in some cases manages to figure it out before reaching step V3, perhaps using information provided by the prover in steps P1 and P2 (of course, such information could only help the verifier by sheer luck since the prover does not know the discrete logarithm of s; nevertheless, such luck is possible and must be taken into account); or

4. the verifier decides at some point to "misbehave", that is to refuse to cooperate in a way that would cause the prover to terminate prematurely the protocol (this happens for instance if the verifier supplies $s = 0$ at step V1, if he claims at step V3 that the prover has cheated with the control blobs when in fact this is not the case, or if he supplies "2" as one of his challenge y_i).

In the first case, the verifier's challenges could depend on the prover's input at steps P1 and P2. If this happens, the simulator will use its resetting capability to force the verifier to disclose the discrete logarithm of s. Once the simulator knows this information, it can "cheat" when it simulates step P3 since this gives it the key to the bit commitment scheme trap-door.

In the second case, the simulator can run the verifier once in order to obtain his challenges at step V3, rewind the verifier, and resimulate step P1 with commitments that it can open to meet the same challenges.

The third and fourth cases are more difficult and will be dealt with by the simulator formally described below. In particular, the fourth case is initially dealt with when we prescribe that the simulator should "output the verifier's view so far (including the random tape) and stop when the verifier refuses to behave properly".

In order to describe the simulator, it is more convenient to think of the verifier as if it were a deterministic process having access to a read-only one-way purely random tape. This random tape is put under the control of the simulator. Initially, this tape is empty. Whenever the verifier wishes to access a new bit on his random tape, the simulator flips a fair coin, places the resulting random bit on the tape, and allows the verifier to proceed. At any time it wishes, the simulator can take a snapshot of the verifier and later restore the verifier in the exact state it had then. When the verifier is reset to a previous state, his read-only head on the random tape is reset to where it was in that state, *but* the other random bits that the simulator may have flipped remain there, possibly to be read again by the verifier. Hence, the verifier reuses the same source of random bits after resetting. The advantage of this approach instead of having the simulator prepare the random tape once and for all ahead of time is that it allows to deal with verifiers that do not have an *a priori* time limit. This is important if we want the simulator to work with any verifier whatsoever. Moreover, it is crucial even if we only consider probabilistic verifiers that run in *expected* polynomial time but that could take significantly more time with small probability.

We now proceed to describe the simulator. When we mention step 0, step Vi or step Pi, we refer to the corresponding steps in Protocol 4. You will notice that the simulation follows exactly Protocol 4 up to and including step S6 (with the simulator in the role of the prover).

Protocol 5 (the simulator)

S: The simulator carries out the initialization step 0 with the verifier exactly as the prover would. If the verifier refuses to behave properly, the simulator outputs the verifier's view so far (including the random tape) and stops.

S1: The simulator waits for the verifier to supply s and $\vec{e}$ as expected in step V1. If the verifier refuses to behave properly, the simulator outputs the verifier's view so far and stops. Otherwise, the simulator takes a snapshot of the verifier.

S2: The simulator proceeds exactly as the prover would in step P1; in particular, it sends the $\vec{x_i}$'s and $\vec{u_{ij}}$'s to the verifier.

S3: The simulator waits for the verifier to supply the $\vec{h_{ij}}$'s as expected in step V2. If the verifier refuses to behave properly, the simulator outputs the verifier's view so far and stops.

S4: The simulator proceeds exactly as the prover would in step P2; in particular, it sends the $\vec{w_{ij}}$'s and the $\vec{t_{ij}}$'s to the verifier.

... **continued on next page** ...

Protocol 5 (continued)

S5: The simulator waits for the verifier to supply $\vec{r}$ and $\vec{y}$ as expected in step V3. If the verifier refuses to behave properly, the simulator outputs the verifier's view so far and stops.

S6: The simulator checks that $\varphi(\vec{e}, \vec{r}, \vec{y})$. If this fails, the simulator outputs the verifier's view so far and stops. Otherwise, the simulator saves $\vec{r}$ and $\vec{y}$ into $\hat{r}$ and $\hat{y}$ for future use.

S7: (This is the turning point in the simulation.) The simulator restores the verifier to its state immediately after step S1.

S8: (The simulator sends commitments that it can open but only if challenges $\hat{y}$ are asked again.) For each i such that $\hat{y}_i = 0$, the simulator computes $\vec{c_i} = \sigma(\Psi)$ and $(\vec{x_i}, \vec{z_i}) = \beta(\vec{c_i})$. For each i such that $\hat{y}_i = 1$, the simulator computes $\vec{d_i} = \rho(\Psi)$ and $(\vec{x_i}, \vec{z_i}) = \beta(\vec{d_i})$. After this, the simulator proceeds exactly as the prover would have at step P1: for each blob $x_i[j]$, the simulator chooses $\vec{b_{ij}} \in \{0,1\}^k$ at random and computes $(\vec{u_{ij}}, \vec{v_{ij}}) = \beta(\vec{b_{ij}})$. It sends all the $\vec{x_i}$'s and $\vec{u_{ij}}$'s to the verifier.

S9: The simulator waits for the verifier to supply the $\vec{h_{ij}}$'s as expected in step V2. If the verifier refuses to behave properly, *the simulator goes back to step S7.*

S10: The simulator proceeds as the prover would in step P2, except that whenever $\hat{y}_i = 1$, $c_i[j]$ should be replaced by $\diamondsuit(d_i[j])$; in particular, it sends the $\vec{w_{ij}}$'s and the $\vec{t_{ij}}$'s to the verifier.

S11: The simulator waits for the verifier to supply $\vec{r}$ and $\vec{y}$ as expected in step V3. It is possible that this response from the verifier will be different from what he had given previously at step S5. If the verifier refuses to behave properly, the simulator goes back to step S7.

... **continued on next page** ...

Protocol 5 (continued)

S12: The simulator checks that $\varphi(\vec{e},\vec{r},\vec{y})$. If this fails, the simulator goes back to step S7.

S13: There are two possibilities at this point: either $\vec{y}=\hat{y}$ or not. In the first case, the simulator had prepared its commitments so that it can meet the challenges. In the second case, this means that the verifier has changed his challenges, which allows the simulator to compute the discrete logarithm of s. If $\vec{y}=\hat{y}$, go to step S15; otherwise, continue.

S14: (The verifier has changed his challenges between steps S5 and S11.) Consider any i, $1 \le i \le k$, such that $y_i \ne \hat{y}_i$. The simulator computes $a = r_i - \hat{r}_i$ if $y_i = 0$, or $a = \hat{r}_i - r_i$ if $y_i = 1$ (once again, do not forget that this subtraction is done in $\mathbf{Z}_{p-1}$, hence modulo $p-1$). Because $\alpha^{r_i} s^{y_i} = e_i = \alpha^{\hat{r}_i} s^{\hat{y}_i}$, it is clear that $\alpha^a = s$, hence the simulator has obtained the discrete logarithm of s.

For each i such that $y_i = 0$ and $\hat{y}_i = 1$, the simulator computes $\vec{c_i} = \sigma(\Psi)$, and for each i such that $y_i = 1$ and $\hat{y}_i = 0$, the simulator computes $\vec{d_i} = \rho(\Psi)$. Now, for each i such that $y_i \ne \hat{y}_i$, the simulator changes the $z_i[j]$'s chosen in step S8 when necessary, as explained below. This will allow the simulator to opens the "commitments" contained in $\vec{x_i}$ in a way that will satisfy challenge y_i even though $\vec{x_i}$ was originally prepared in order to satisfy challenge $\hat{y}_i$. This is done as follows. For each i such that $y_i \ne \hat{y}_i$ and for each j such that $c_i[j] \ne \Diamond(d_i[j])$,

- if $\hat{y}_i = c_i[j]$, the value of $z_i[j]$ is decremented by a;
- if $\hat{y}_i \ne c_i[j]$, the value of $z_i[j]$ is incremented by a.

S15: (The simulator is now ready to output the verifier's view.) The simulator outputs the verifier's random tape, the $\vec{x_i}$'s and the $\vec{u_{ij}}$'s from step S8, and the $\vec{w_{ij}}$'s and the $\vec{t_{ij}}$'s from step S10. Moreover, For each i such that $y[i] = 0$, the simulator outputs $\vec{z_i}$ and $\vec{c_i}$; for each i such that $y[i] = 1$, the simulator resets $z_i[j]$ to 0 when $d_i[j] = \square$, and it outputs $\vec{z_i}$ and $\vec{d_i}$. The simulator halts.

8.2 The simulator produces the correct distribution

Let Ψ be a satisfiable boolean expression. Let $\vec{a}$ be a satisfying assignment for Ψ known by the honest prover. Let ℓ denote $\ell(\Psi)$, the length of each member of $\mathcal{C}_\Psi$ and $\mathcal{D}_\Psi$. Consider any verifier. Consider also an infinite string R of random bits. From now on, consider Ψ, $\vec{a}$, ℓ, R, and the verifier as fixed. Hence, when we say that "A is determined uniquely by B", we mean "by B and these fixed parameters".

Let us analyse the view of the verifier supplied with R on his random tape when interacting with the honest prover prescribed by Protocol 4, and compare it with the view produced by the simulator when it supplies the appropriate initial segment of the same R to the verifier on his random tape. For simplicity, let us assume that the verifier does not misbehave so early that step P1 is not even entered; otherwise, it is obvious that the simulation is perfect. For simplicity, we shall also pretend that step 0 in Protocol 4 and step S0 in Protocol 5 do not exist, because it is clear that step S0 simulates exactly step 0.

Now that we have excluded step 0 from consideration, observe that the verifier's s and $\vec{e}$ supplied at step V1 are determined by the fixed parameters, and thus may as well be considered fixed too. Observe also that the *only* step of Protocol 4 in which the prover makes random choices is step P1. Hence, everything in Protocol 4 is determined entirely by the prover's choices of the $\vec{c_i}$'s, $\vec{z_i}$'s, $\vec{b_{ij}}$'s, and $\vec{v_{ij}}$'s. (The $\vec{x_i}$'s and $\vec{u_{ij}}$'s are determined from these random choices.)

More to the point, the verifier's view of Protocol 4 until just before step P3, when interacting with the prover, is determined entirely by the $\vec{x_i}$'s, $\vec{u_{ij}}$'s, and $\vec{t_{ij}}$'s supplied by the prover at steps P1 and P2 (unless step P2 does not take place, in which case the verifier's view is determined by the $\vec{x_i}$'s and $\vec{u_{ij}}$'s alone). This is true because the $\vec{w_{ij}}$'s supplied by the honest prover at step P2 (assuming it takes place) are determined uniquely by the $\vec{x_i}$'s, $\vec{u_{ij}}$'s, $\vec{t_{ij}}$'s, and by the $\vec{h_{ij}}$'s supplied by the verifier at step V2, but the $\vec{h_{ij}}$'s themselves are determined uniquely by the $\vec{x_i}$'s and the $\vec{u_{ij}}$'s. To see this, notice that the tests performed by the honest verifier at step V3 give no freedom for the $\vec{w_{ij}}$'s that will not cause him to reject, because the discrete logarithm is uniquely defined.

A crucial observation is that the prover's protocol results in each $\vec{x_i}[j]$, $\vec{u_{ij}}[m]$ and $\vec{t_{ij}}[m]$ being produced randomly and independently according to the uniform distribution over $\mathbf{Z}_p^*$, $\mathbf{Z}_p^*$, and $\{0,1\}$, respectively (except that the $\vec{t_{ij}}$'s are not produced when step P2 is not entered). Let Λ denote the space of possible values for the collection of $\vec{x_i}$'s, $\vec{u_{ij}}$'s, and $\vec{t_{ij}}$'s. More precisely, $\Lambda = (\mathbf{Z}_p^* \times (\mathbf{Z}_p^* \times \{0,1\})^k)^{\ell k}$. Consider any element $\lambda \in \Lambda$. We say that λ is *good* if Protocol 4 reaches step V4 when the verifier is supplied the view consistent with λ at steps P1 and P2. Otherwise, λ is *bad*. ("Consistent" means that the $\vec{w_{ij}}$'s take the unique values that would not cause the honest verifier to reject at step V3.) Finally, let $\wp$ denote the number of good elements in Λ divided by the total number of elements in Λ. In simple terms, $\wp$ stands for the probability that the verifier does not misbehave in Protocol 4, the probability being taken over the random choices of the prover (we do not care if the verifier misbehaves in step V4 by rejecting when he should not, because this decision is not part of his view). Of course, $\wp$ may be a function of the random tape R, but this does not concern us here since R is fixed.

The interaction of the verifier with the honest prover in Protocol 4 is characterized as follows. The prover chooses the $\vec{c_i}$'s, $\vec{z_i}$'s, $\vec{b_{ij}}$'s, and $\vec{v_{ij}}$'s randomly at step P1. This determines the $\vec{x_i}$'s and $\vec{u_{ij}}$'s. If the verifier misbehaves at step V2, consider random binary values for the $\vec{t_{ij}}$'s; otherwise, the prover's $\vec{x_i}$'s and $\vec{u_{ij}}$'s determine the verifier's $\vec{h_{ij}}$'s, which in turn determine the prover's $\vec{t_{ij}}$'s at step P2. Let λ stand for the collection of $\vec{x_i}$'s, $\vec{w_{ij}}$'s, and $\vec{t_{ij}}$'s thus produced. Once again, λ is uniformly distributed over Λ by the prover's random choices at step P1.

If λ is good, which happens with probability $\wp$, it determines the verifier's challenge vector $\vec{y}$ at step V3. At step P3, the prover shows the verifier $\vec{c_i}$'s in $\mathcal{C}_\Psi$ when $y_i = 0$ and and $\vec{d_i}$'s in $\mathcal{D}_\Psi$ when $y_i = 1$, together with the appropriate $\vec{z_i}$'s. Even though the $\vec{c_i}$'s and $\vec{d_i}$'s were determined as early as step P1, they are completely uncorrelated with λ because blobs are uncorrelated with the bits they hide. Moreover, the $\vec{c_i}$'s are uniformly distributed over $\mathcal{C}_\Psi$ (because they are produced by a call on $\sigma(\Psi)$ at step P1), and the $\vec{d_i}$'s are uniformly distributed over $\mathcal{D}_\Psi$ (because they are produced as $\gamma(\Psi, \sigma(\Psi), \vec{a})$ in steps P1 and P3). Quite the opposite, the $\vec{z_i}$'s are determined completely by λ, the $\vec{c_i}$'s (when $h_i = 0$) and the $\vec{d_i}$'s (when $h_i = 1$).

In conclusion, the verifier's view of Protocol 4 is uniquely determined by a random λ uniformly distributed over Λ and, if λ is good, by a collection of random $\vec{c_i}$'s and $\vec{d_i}$'s uniformly distributed over $\mathcal{C}_\Psi$ and $\mathcal{D}_\Psi$, respectively.

Let us now consider the view produced by the simulator in Protocol 5 when it supplies the verifier with the appropriate initial segment of the same random tape R. Clearly, Protocol 5 mimics exactly Protocol 4 until (and if) it reaches step S7. Therefore, with probability $1-\wp$, the simulation never reaches step S7 and it outputs the view of a verifier that misbehaves. This view is produced according to the same probability distribution than if the verifier had interacted with the prover. Moreover, this view is consistent with a $\lambda \in \Lambda$ that is not good, and this λ is obtained according to the uniform distribution over the bad elements of Λ (assuming that $\wp \neq 1$).

On the other hand, the simulation reaches step S7 with probability $\wp$. This happens if and only if the view of the verifier so far is consistent with a good $\lambda \in \Lambda$, in which case this λ is distributed uniformly over the good elements in Λ. Each attempt to go from step S7 to step S13 can be thought of as sampling within Λ uniformly in the hope of hitting a good element again. Clearly, whenever such a good element λ is reached, it is obtained according to the uniform distribution over the good elements of Λ.

From there, step S15 will produce $\vec{c_i}$'s in $\mathcal{C}_\Psi$ when the verifier's challenge y_i is 0 and $\vec{d_i}$'s in $\mathcal{D}_\Psi$ when y_i is 1, together with the unique $\vec{z_i}$'s consistent with the $\vec{x_i}$'s in λ. Regardless of whether these $\vec{c_i}$'s and $\vec{d_i}$'s were chosen at step S8 or S14, they are uniformly distributed over $\mathcal{C}_\Psi$ (by a call on $\sigma(\Psi)$) and $\mathcal{D}_\Psi$ (by a call on $\rho(\Psi)$) in a way that is uncorrelated with λ.

To summarize, the view produced by the simulator is consistent with a random λ uniformly distributed over the bad elements of Λ with probability $1-\wp$, or with a random λ uniformly distributed over the good elements of Λ with probability $\wp$. This is just a complicated way of saying that the view produced by the simulator is consistent with a random λ uniformly distributed over all the elements of Λ. Moreover, if λ is good, this view includes random $\vec{c_i}$'s and $\vec{d_i}$'s (as required by the h_i's consistent with λ) uniformly distributed over $\mathcal{C}_\Psi$ and $\mathcal{D}_\Psi$, together with the appropriate $\vec{z_i}$'s. In other words, the view produced by the simulator is distributed exactly as the verifier's view when interacting with the honest prover. The two key properties of the basic BCC–protocol that make all this work are that both $\rho(\Psi)$ and $\gamma(\Psi, \sigma(\Psi), \vec{a})$ produce elements of $\mathcal{D}_\Psi$ according to the same probability distribution, and that blobs are information-theoretically uncorrelated with the bits they hide. This concludes the proof that the simulation is perfect.

8.3 The simulator is efficient

Let $\wp$ and Λ be as in Section **??**, as well as all the fixed parameters. Clearly, $\wp$ is exactly the probability that the simulator reaches step S7 in Protocol 5 because until then Protocol 5 mimics precisely Protocol 4. More interestingly, $\wp$ is also exactly the probability that the simulator, if in step S7, can reach step S13 (and hence terminate happily at step S15) without going back at least once to step S7. In order to see this, observe again that the collection of $\vec{x_i}$'s, $\vec{u_{ij}}$'s, and $\vec{t_{ij}}$'s obtained in steps S8 and S10 is produced randomly according to the uniform distribution over Λ.

Consequently, a simple calculation shows that the expected number of times that step S7 is entered is exactly 1 (unless $\wp = 0$, in which case this number is 0). But it is clear that steps S0 to S6 cannot be entered more than once each, and that none of the steps after S7 can be entered more times than step S7 itself. Therefore, the expected time for the simulator's protocol is no more than the expected time for executing once each of its steps, which is efficient with respect to the verifier's efficiency.

This completes the proof that Protocol 4 is perfect zero-knowledge.

9 One-way certified group actions

Our final protocol, as given in Section **??** is computationally convincing under the certified discrete logarithm assumption (Section **??**). Clearly, a weaker assumption is sufficient to make it work: all we need is a trap-door bit commitment scheme unconditionally secure for the originator, such that EPR–blobs can be detected except possibly with exponentially small probability.

It would be very nice if such bit commitment schemes could be designed under the sole assumption that one-way functions exist. Although we are not able to do this, we now proceed to show that it is sufficient to assume the existence of *one-way certified group actions*, which is a new notion introduced in this paper. This is a generalization of the *one-way group homomorphism* introduced in [IY] and used in a previous version of the current paper [BCY].

9.1 Definitions

First of all, let us recall what a group action is. Let G be a finite group whose operation is simply denoted by juxtaposition and let ε denote the identity element of G. Let S be any finite set. We say that G *acts* on S if each element of G induces a permutation of S such that the permutation induced by gh is the composition of the permutations induced by g and by h, where g and h are any elements of G. More formally, we have a function $T : G \to (S \to S)$ such that

(1) $(\forall g \in G)(\forall h \in G)(\forall s \in S)[(T(gh))(s) = (T(g))((T(h))(s))]$, and

(2) $(\forall g \in G)(\forall t \in S)(\exists! s \in S)[(T(g))(s) = t]$. (It is an easy exercise to show that this unique s is $(T(g^{-1}))(t)$.)

Given condition (1), it is elementary to prove that condition (2) is equivalent to saying that the function induced by the group identity element is the identity function. In other words, G act on S through T if and only if conditions (1) above and (3) below are satisfied.

(3) $(\forall s \in S)[(T(\varepsilon))(s) = s]$.

To avoid cluttering the text with parentheses, it is customary to denote $(T(g))(s)$ simply by gs. Therefore, condition (1) can be restated simply as $(gh)s = g(hs)$. (Despite the appearance of this formula, it does not really have anything to do with associativity!)

Let us now suppose that G acts on S. Let s_0 be a fixed element of S. The group action is s_0–*one-way* if

- Membership in G and S can be tested efficiently.
- It is feasible to draw randomly within G with uniform distribution. By $g \in_R G$, we mean that g is chosen randomly within G with uniform distribution.
- The group operation, the group inversion, and the group action can be computed efficiently. In other words, given any $g \in G$, $h \in G$ and $s \in S$, it is easy to compute gh, g^{-1}, and gs.

- Consider $g \in_R G$ and let $t = gs_0$. Given s_0 and t, it is infeasible to compute any $\hat{g}$ such that $\hat{g}s_0 = t$, except with negligible probability, where the probability is taken over all choices of g and possibly over the random choices taken by the efficient algorithm trying to defeat this property. Note that the problem is *not* to find $\hat{g} \neq g$.

A group action is *one-way* if it is feasible to find an $s_0 \in S$ such that the group action is s_0–one-way. Such an s_0 will be referred to as the *source* of the action. Given any $s \in S$, let $Q(s)$ denote $\{t \in S \mid (\exists g \in G)[t = gs]\}$. A one-way group action whose source is s_0 is *certified* if

- Given any $t \in S$, it is easy to decide whether or not $t \in Q(s_0)$. Of course, the easy thing is to decide on the *existence* of a $g \in G$ such that $t = gs_0$, not to actually *discover* any such g.

A one-way group action whose source is s_0 is *statistically* certified if

- Given any $t \in S$, it is easy to "decide" whether or not $t \in Q(s_0)$, with an arbitrarily small error probability. As a slight relaxation of this notion, we allow the party who wishes to verify that $t \in Q(s_0)$ to seek help from another party who may know a $g \in G$ such that $t = gs_0$. This other party, however, should not help the initial party to find any such g and it should not require an unbounded number of rounds.

9.2 Commitments with group actions

Let G be a group and S be a set, and consider a one-way certified group action.

- In order to set up a bit commitment scheme, the receiver chooses a source s_0 for the action and a $g_0 \in_R G$. He computes $s_1 = g_0 s_0$ and he gives s_0 and s_1 to the originator. The originator checks that $s_0 \in S$, $s_1 \in S$, and $s_1 \in Q(s_0)$.
- In order to commit to bit $x \in \{0, 1\}$, the originator chooses $g \in_R G$ and computes $b = gs_x$. She keeps g as her witness to the effect that b is an x–blob.
- In order to open a commitment b as bit x, the originator shows the corresponding witness g. The receiver checks that $b = gs_x$.

Notice that 0–blobs are produced by computing gs_0 whereas 1-blobs are produced by computing $gs_1 = g(g_0s_0) = (gg_0)s_0$, where $g \in_R G$. Therefore, such commitments are information-theoretically secure for the originator because the effect of computing gg_0 for a fixed $g_0 \in G$ and a g chosen randomly with uniform distribution within G is in fact to choose randomly an element of G with uniform distribution. In other words, nothing distinguishes a 0–blob from a 1–blob, except for the witness known by the originator alone. The condition that the one-way group action should be certified is *crucial* here: if the receiver were able to get away with giving the originator some $s_1 \notin Q(s_0)$, the set of 0–blobs would be *disjoint* from the set of 1–blobs!

On the other hand, assume for a contradiction that the originator is able to open a given blob b both as a 0–blob and as a 1–blob. In order to do this, she must know x_0 and x_1 in G such that $b = x_is_i$. But then $x_1^{-1}x_0s_0 = x_1^{-1}x_1s_1 = s_1$. If the originator computes $g = x_1^{-1}x_0$, she will have found a $g \in G$ such that $gs_0 = s_1$, which is precisely what was assumed to be infeasible by the one-wayness of the action.

9.3 How to use such commitments

Bit commitment schemes based on one-way certified group actions can be used directly in the basic BCC–protocol. However, there is a technical difficulty if you want to use them in the constant-round protocol given in Section **??**: there is no obvious way to implement control blobs as described in Section **??** in order to defeat the EPR threat!

Fortunately, Benaloh's original cryptographic capsules [Be] can be used instead of control blobs at the cost of only doubling the number of blobs involved in the protocol. This is possible because bit commitment schemes based on one-way certified group actions allow the originator of two blobs b_1 and b_2 to convince the receiver that they are commitments to the same bit (unless they are both EPR) if in fact this is so. In order to do so, let $x \in \{0, 1\}$ and let g_1 and g_2 be the originator's witnesses for blobs b_1 and b_2, respectively. If the originator provides the receiver with $h = g_2g_1^{-1}$, the receiver can check that $hb_1 = b_2$.

We leave it for the reader to verify that the only way the originator can provide an $h \in G$ that transforms b_1 into b_2 is if she can open both blobs to show the same bit, unless both blobs are EPR. We also leave it for the reader to verify that no information about which way both these blobs can be opened leaks when the originator gives h to the receiver. (The reason why control blobs cannot be implemented directly with one-way certified group actions is that they do not provide a natural mechanism by which the originator can convince the receiver that he can open blobs b_1 and b_2 to show *different* bits, assuming they are not EPR.)

Bit commitment schemes based on one-way *statistically* certified one-way group actions can also be used in the constant-round protocol of in Section **??**, but the resulting computationally convincing protocol will merely be statistically zero-knowledge rather than perfect zero-knowledge [GMW].

9.4 Examples of one-way certified group actions

9.4.1 Back to the discrete logarithm

Assuming the certified discrete logarithm assumption, a one-way certified group action can be built as follows. Let p be a prime for which the factorization of $p-1$ is known and let α be a generator for $\mathbf{Z}_p^*$. Let G be $\mathbf{Z}_{p-1}$, let S be $\mathbf{Z}_p^*$, and let s_0 be 1. Given $g \in G$ and $s \in S$, the group action is defined as $gs = s\alpha^g$. It is easy to see that all the requirements for a one-way certified group action are satisfied. In this case, the one-wayness of the group action follows directly from the certified discrete logarithm assumption and the one-way group action is certified because recognizing elements of $Q(s_0)$ is trivial since $S = Q(s_0)$ follows from the fact that α is a certified generator.

9.4.2 One-way group homomorphisms

Consider any one-way group homomorphism $h : X \to Y$ (see [BCY, Section 5] for a definition) such that membership in X and Y can be tested efficiently (an important condition forgotten in [BCY]). Let G be X, S be Y, and s_0 be the identity element of Y. Given $g \in G$ and $s \in S$, the group action is defined as $gs = s * h(g)$. Details that this defines a one-way certified group action are left for the reader.

9.4.3 Scheme based on graph isomorphism

The notion of one-way group homomorphisms described in [BCY] provided a generalization of the bit commitment scheme based on the certified discrete logarithm assumption, but it was probably not as general as one-way certified group actions. Although we do not know any one-way *certified* group action that could not be recast in the context of one-way group homomorphisms, we now describe a one-way *statistically* certified group action that does not correspond to a one-way group homomorphism. The one-wayness of our group action depends on an unproved assumption introduced in [BC2].

Let n be a fixed large integer. Let G be the group of permutations of $X_n = \{1, 2, \ldots, n\}$ under composition (where $(gh)(i) = h(g(i))$). Let S be the set of all graphs with X_n as vertex set. Let us assume the existence of a *hard graph* in the sense of [BC2]: a graph is hard if it is infeasible to figure out an isomorphism between it and a random isomorphic copy of it, except with negligible probability. Let $s_0 \in S$ be such a hard graph. Given $g \in G$ and $s = (X_n, E) \in S$, the group action is defined as $gs = (X_n, \hat{E})$, where $(u, v) \in \hat{E}$ if and only if $(g(u), g(v)) \in E$. This group action is one-way by assumption and it is statistically certified because of the protocol given in [BMO]. Intuitively, the reason why this group action cannot be recast as a group homomorphism is that there is no natural group operation that one could put on S.

Acknowledgements

It is a pleasure to acknowledge fruitful discussions with Charles H. Bennett, Joan Boyar, David Chaum, Ivan Damgård, Uri Feige, Oded Goldreich, Shafi Goldwasser, Joe Kilian, Phillip Rogaway, and Adi Shamir.

References

[AGR] Aspect, A., Grangier, P. and Roger, G., "Experimental realization of the Einstein–Podolsky–Rosen–Bohm Gedankenexperiment: A new violation of Bell's inequalities", *Physical Review Letters*, Vol. 49, 1982, pp. 91 – 94.

[BMO] Bellare, M., Micali. S. and Ostrovsky, R., "Perfect zero-knowledge in constant rounds", *Proceedings of the 22nd ACM Symposium on Theory of Computing*, 1990, to appear.

[Be] Benaloh, J. C., "Cryptographic capsules: A disjunctive primitive for interactive protocols" *Advances in Cryptology: CRYPTO '86 Proceedings*, Springer–Verlag, 1987, pp. 213 – 222.

[BB1] Bennett, C. H. and Brassard, G., "Quantum cryptography: Public-key distribution and coin-tossing", *Proceedings of the International Conference on Computers, Systems & Signal Processing*, Bangalore, India, 1984, pp. 175 – 179.

[BB2] Bennett, C. H. and Brassard, G., "Quantum cryptography", in preparation.

[BM] Blum, M. and Micali, S., "How to generate cryptographically strong sequences of pseudo-random bits", *SIAM Journal on Computing*, Vol. 13, 1984, pp. 850 – 864.

[BKK] Boyar, J. F., Krentel, M. W. and Kurtz, S. A., "A discrete logarithm implementation of zero-knowledge blobs", *Journal of Cryptology*, to appear.

[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., "Zero-knowledge simulation of Boolean circuits", *Advances in Cryptology: CRYPTO '86 Proceedings*, Springer–Verlag, 1987, pp. 224 – 233.

[BC2] 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.

[BC3] Brassard, G. and Crépeau, C., "Sorting out zero-knowledge", *Advances in Cryptology: EUROCRYPT '89 Proceedings*, Springer–Verlag, 1990, 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.

[Ch] 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.

[CDG] Chaum, D., Damgård, I. B. and van de Graaf, J., "Multiparty computations ensuring privacy of each party's input and correctness of the result", *Advances in Cryptology: CRYPTO '87 Proceedings*, Springer–Verlag, 1988, pp. 87 – 119.

[CEG] Chaum, D., Evertse, J.–H. and van de Graaf, J., "An improved protocol for demonstrating possession of discrete logarithms and some generalizations", *Advances in Cryptology: EUROCRYPT '87 Proceedings*, Springer–Verlag, 1988, pp. 127 – 141.

[CEGP] Chaum, D., Evertse, J.–H., van de Graaf, J. and Peralta, R., "Demonstrating possession of a discrete logarithm without revealing it", *Advances in Cryptology: CRYPTO '86 Proceedings*, Springer–Verlag, 1987, pp. 200 – 212.

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

[FS] Feige, U. and Shamir, A., "Zero knowledge proofs of knowledge in two rounds", *Advances in Cryptology: CRYPTO '89 Proceedings*, Springer–Verlag, 1990, to appear.

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

[Gka] Goldreich, O. and Kahn, A., "Using claw–free permutations to construct zero-knowledge proofs for **NP**", in preparation.

[GKr] Goldreich, O. and Krawczyk, H., "On the composition of zero-knowledge proof systems", 1989, manuscript.

[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.

[IY] Impagliazzo, R. and Yung, M., "Direct minimum-knowledge computations", *Advances in Cryptology: CRYPTO '87 Proceedings*, Springer–Verlag, 1988, pp. 40 – 51.

[K] Kranakis, E., *Primality and Cryptography*, Wiley–Teubner Series in Computer Science, 1986.

[M] Maurer, U. M. "Fast generation of secure RSA–moduli with almost maximal diversity", *Advances in Cryptology: EUROCRYPT '89 Proceedings*, Springer–Verlag, 1990, to appear.

[NY] Naor, M. and Yung, M., "Universal one-way hash functions and their cryptographic applications", *Proceedings of the 21st ACM Symposium on Theory of Computing*, 1989, pp. 33 – 43.

[PH] Pohlig, S. and Hellman, M. E., "An improved algorithm for computing logarithms over $\mathbf{GF}(p)$ and its cryptographic significance", *IEEE Transactions on Information Theory*, Vol. IT–24, 1978, pp. 106 – 110.

[S] Shannon, C. E., "A mathematical theory of communications", *Bell System Technical Journal*, Vol. 27, 1948, pp. 379 – 423 and 623 – 656.