Everything in NP can be argued in *perfect* zero-knowledge in a *bounded* number of rounds

Gilles BRASSARD [†]

Département IRO
Université de Montréal

Claude CREPEAU [‡]

Laboratory for Computer Science
Massachusetts Institute of Technology

Moti YUNG

IBM T. J. Watson Research Center
Yorktown Heights, NY

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 (under the assumption that it is possible 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 $p-1$, or more generally under the assumption that one-way group homomorphisms exist). All these protocols are BCC-*arguments* rather than GMR-proofs [BC3].

[†] Supported in part by Canada NSERC grant A4107.

[‡] Supported in part by an NSERC postgraduate scholarship; part of this research was performed while this author was visiting the IBM Almaden Research Center.

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 proofs (under the assumption that secure encryption functions exist) [GMW]. See also [BC1]. Such proofs, 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). 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]. We shall refer to the settings of either [Ch] or [BC2] as the BCC-setting in order to contrast it with the GMR-setting described in the previous paragraph [BC3].

The difference between these settings 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. after completion of the protocol.) 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 the same three authors [BCC], thanks to an idea of Damgaard, yielding in effect a *perfect* zero-knowledge protocol in the terminology of [GMW], but in the BCC-setting. This implies that the prover's safety would still be guaranteed in 500 years, even if strong organizations with unknown computing power and algorithmic knowledge were to try to extract her secret. To summarize, 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).

Unfortunately, Fortnow proved that there can be no free lunch (unless the polynomial hierarchy collapses) [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. This is why the results of [Ch, BC2, BCC] could presumably not have been obtained in the GMR-setting. More importantly in practice, even a polynomial-time prover can cheat these protocols if the cryptographic assumptions turn out to be ill-founded, but the prover's cheating must be performed while the protocol is taking place. For this reason, protocols in the BCC-setting are better called *arguments* rather than proofs [BC3]. Read [BCC] for additional comparison between these settings and results, for the related notion of *minimum disclosure*, and for further discussion.

The main motivation behind the work of [GMW, BC1, Ch, BC2, BCC] described above was a quest for *generality*: how much is it possible to prove in zero-knowledge 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 could not 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. In sharp contrast, the more general protocols of [GMW, BC1, Ch, BC2, BCC] require an unbounded number of rounds in order to achieve an arbitrarily high level of safety. This paper addresses the following question: Is it possible to combine generality and arbitrarily high safety with a small (constant?) number of rounds? (By one ''round'', we mean two ''moves'': a message sent by the verifier followed by a message sent by the prover.) Our answer is that three rounds suffice (under the assumption that it is possible 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 $p-1$, or more generally under the assumption that one-way group homomorphisms exist — see section 5).

A similar question is currently being investigated independently by other researchers. In the GMR-setting, Goldreich claims a bounded-round computational zero-knowledge protocol for all statements in **NP** [G]. Feige and Shamir have also developed a bounded-round computational zero-knowledge protocol for all statement in **NP**, but in a setting in which both the prover and the verifier are limited to probabilistic polynomial time [FS]. However, being merely *computational* zero-knowledge, neither of these protocols offer unconditional safety for the prover. In this paper, we give the first published bounded-round *perfect* zero-knowledge protocol for all statements in **NP** (in the BCC-setting). Note that Feige and Shamir also claim in [FS] that they have a bounded-round (in fact 2 rounds) perfect zero-knowledge protocol, but they give no detail in the current version of their paper (March 1989). Nevertheless, it is clear that their solution is significantly different from ours.

2. Towards the protocol

The protocols of [GMW, BC1, Ch, BC2, BCC] all follow the same basic template, which we call the ‘‘sequential’’ approach. Schematically:

```
{perhaps some initialization}
{let $k$ be a safety parameter}
for i ← 1 to $k$ do
  (1) The prover commits herself by sending $x_i$ to the verifier.
  (2) The verifier sends challenge $y_i$ to the prover.
  (3) The prover meets the challenge by sending $z_i$ to the verifier.
```

At step 1, 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, 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 with each round. In fact, the verifier's confidence grows exponentially with the number k of rounds. We assume here that the reader is already familiar with this type of reasoning, which is the basic principle of all zero-knowledge and minimum disclosure protocols; otherwise, we suggest he/she should read [BCC] first.

In order to reduce the number of rounds to a constant, the following template, which we call the ‘‘parallel’’ approach, comes immediately to mind:

```
{perhaps some initialization}
{let $k$ be a safety parameter}
(1) The prover commits herself by sending $x_1, x_2, \ldots, x_k$ to the verifier.
(2) The verifier sends challenges $y_1, y_2, \ldots, y_k$ to the prover.
(3) 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 if this choice of challenges is made according to a one-way 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 her secret information). Whether this could in some cases help the verifier compute the prover's secret is a most interesting open question. Read [BCC] for a discussion of this issue.

In order to sketch our perfect zero-knowledge parallel solution (in the BCC-setting), we must review a bit commitment scheme used in [BCC] (due in part to Damgaard [CDG] and independently to Boyar, Krentel and Kurtz [BKK]). The purpose of a bit commitment scheme is to allow one party (usually the prover) to commit to the value of a bit in a way that prevents the other party from learning it without the first party's help, but also in a way that prevents the first party from changing its value.

Initially, the prover and the verifier agree on a large prime p for which they both know the factorization of $p-1$ (finding large primes p with known factorization of $p-1$ and with a known generator can be done efficiently in practice). They also agree on a generator α for $\mathbb{Z}_p^*$, the multiplicative group of integers modulo 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 $\mathbb{Z}_p^*$. Given any integer i, $0 \le i \le p-2$, it is easy to compute $\alpha^i \ \mathbf{mod}\ p$ efficiently, 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*. In order to set up a bit commitment scheme, the verifier chooses a random $s \in \mathbb{Z}_p^*$ and gives it to the prover. We assume that the prover is not capable of computing the discrete logarithm of s while the protocol is in progress.

In order to commit to bit $x \in \{0,1\}$, the prover selects a random integer r, $0 \le r \le p-2$, and she computes $b = \alpha^r s^x \ \mathbf{mod}\ p$. She gives b to the verifier but she keeps r as her secret *witness*. For convenience, we shall refer to b as a *blob*. Subsequently, if the prover wishes to show the verifier which bit she had committed to with a given blob b, an operation known as *opening* the blob, she simply shows him the corresponding witness r. The verifier can then check whether $b \equiv \alpha^r \pmod{p}$ or $b \equiv \alpha^r s \pmod{p}$.

It is easy to see [BCC] that the prover is able to open a given blob both ways (i.e. she is able to cheat) if and only if she can efficiently compute (or already knows) the discrete logarithm of s, which we assumed to be infeasible for her. Moreover, any element of $\mathbb{Z}_p^*$ can be used by the prover as commitment to 0 just as well as to 1, depending only on her knowledge about it (the witness). Therefore, it is information-theoretically impossible for the verifier to distinguish a commitment to 0 from a commitment to 1 (regardless of the verifier's computing power).

Using this bit commitment scheme, a perfect zero-knowledge argumentative protocol for satisfiability is given in [BCC]. (Such a protocol was erroneously claimed in [BC2].) For the protocol of [BCC] to be perfect zero-knowledge, it is crucial that it be performed sequentially. Curiously, however, the parallel version of this argument *is* perfect zero-knowledge under the condition that the verifier knows the discrete logarithm of s. If he is willing to cooperate, this is easy for the verifier to achieve: instead of choosing s at random among $\mathbb{Z}_p^*$, he should choose j at random, $0 \le j \le p-2$, and compute

$s = \alpha^j$ **mod** p. Of course, the difficulty is that the prover has no reason to trust that the verifier will choose s according to this new rule. This is a serious consideration because [BCC] describes a way for the verifier to choose s so that a parallel session of the protocol would probably yield a transcript he could not have obtained by himself. In order to remove this difficulty, the following ‘‘parallel’’ protocol comes to mind:

{initialization}
(1) The prover and the verifier agree on p, α and k.
(2) The verifier chooses an integer j, $0 \le j \le p-2$;
computes $s = \alpha^j$ **mod** p;
sends s to the prover;
convinces the prover that he knows the discrete logarithm of s.
{main protocol as in [BCC] but in parallel}
(3) The prover commits herself by sending $x_1, x_2, \ldots, x_k$ to the verifier.
(4) The verifier sends challenges $y_1, y_2, \ldots, y_k$ to the prover.
(5) The prover meets the challenges by sending $z_1, z_2, \ldots, z_k$ to the verifier.

This looks great... except that all the known protocols by which the verifier could convince the prover that he knows the discrete logarithm of s [CEG, CEGP] are inherently sequential! Although it is generally believed that a parallel version of these protocols would not help the verifying party (who is our prover in this case) in attempts to compute the discrete logarithm of s, no one has yet been able to prove this.

Another idea to obtain an argument that remains perfect zero-knowledge even if run in parallel is to force the verifier to choose his challenges before he gets to see the prover’s commitments. In other words, we could ask the verifier to *commit* to his challenges. The resulting protocol would look like this:

{initialization}
(1) The prover and the verifier agree on p, α and k.
(2) The verifier chooses s in $\mathbb{Z}_p^*$;
sends s to the prover.
{main protocol}
(3) The verifier chooses bits $y_1, y_2, \ldots, y_k$ and commits to them.
(4) The prover commits herself by sending $x_1, x_2, \ldots, x_k$ to the verifier.
(5) The verifier opens challenges $y_1, y_2, \ldots, y_k$ for the prover.
(6) The prover meets the challenges by sending $z_1, z_2, \ldots, z_k$ to the verifier.

The natural question at this point is: ‘‘which bit commitment scheme should the verifier use in step 3?’’ The obvious answer would be to require that the prover also chooses an $\hat{s}$ in $\mathbb{Z}_p^*$, and sends it to the verifier, so that the verifier would commit to bit y by computing $\alpha^r \hat{s}^y$ **mod** p for a random r. However, there is a nicer (and somewhat surprising) solution: the verifier commits to his challenges using his very own s! This sounds crazy because nothing prevents the verifier from knowing the discrete logarithm of s, in which case his ‘‘commitments’’ would be worthless. Nevertheless, the only way the verifier can cheat in his challenge commitments is by knowing the discrete logarithm of s, but this is precisely the case in which the protocol of [BCC] remains perfect zero-knowledge even in parallel, regardless of how or when the verifier chooses his challenges. Therefore, either the verifier chooses s in step 2 in a way that he does not know its discrete logarithm, in which case he must choose his challenges in step 3 before seeing anything from the prover, or he chooses s in a way that he knows its discrete logarithm (i.e. he chooses the discrete logarithm and computes s from it), in which case no harm is done if the verifier does not choose his challenges until step 5.

Unfortunately, the protocol we have just described *still* does not work for a rather subtle reason: it becomes possible for the prover to cheat without finding 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 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 5, she 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 3 by choosing a random r and computing $c = \alpha^r s^y$. After the verifier has given his commitment c to the prover, the prover can create a blob d by computing $d = \alpha^q c \textbf{ mod } p$ for a randomly chosen q. Clearly, the prover cannot open this blob yet. Nevertheless, the verifier will reveal r to the prover in step 5, and this will enable the prover to ‘‘open’’ her blob d as bit y by showing ‘‘witness’’ $q+r \textbf{ mod } p-1$. This ability to relate the value of some of her blobs to the challenges she will subsequently be issued spells doom on the protocol of [BCC]. (This difficulty can only be replaced by other difficulties if we insist that the verifier’s commitments to his challenges be based on a different $\hat{s}$. These new difficulties range from the argument being no longer perfect zero-knowledge to providing other back doors for the prover to cheat.)

In order to solve this ultimate difficulty and finally obtain the promised bounded-round perfect zero-knowledge argumentative protocol, we must force the prover to emit only blobs that she knows how to open at the time of their creation. Let b be a blob created by the prover. We say that it is a 0-blob (resp. a 1-blob) if the prover can exhibit an r such that $b \equiv \alpha^r \pmod{p}$ (resp. $b \equiv \alpha^r s \pmod{p}$). As argued previously, no prover-created blob can be simultaneously a 0-blob and a 1-blob unless the prover knows (or can find 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 Einstein-Podolsky-Rosen ‘‘paradox’’ of quantum physics. 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 subprotocol allows the prover to convince the verifier that a given blob is not EPR.

Let b be a non-EPR-blob, which we shall call the *actual* blob. Let r be the prover's witness that would allow her to open this blob. In order to convince the verifier that she can open blob b, she creates k additional *control* blobs $b_1, b_2, \ldots, b_k$ by randomly selecting k integers $r_1, r_2, \ldots, r_k$, $0 \le r_i \le p-2$, and k bits $x_1, x_2, \ldots, x_k$, and computing $b_i = \alpha^{r_i} s^{x_i}$ **mod** p, $1 \le i \le k$. She gives all the 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 the control blob b_i by showing r_i. For each $h_i = 1$, the prover shows that she can open the actual blob b if and only if she can open the control blob b_i. This is done as follows:

– if b is an x_i-blob, the prover computes $w = r_i - r$ **mod** $p-1$ and gives it to the verifier, who checks that $b_i \equiv \alpha^w b \pmod{p}$;

– if b is a $(1-x_i)$-blob, the prover computes $w = r_i + r$ **mod** $p-1$ and gives it to the verifier, who checks that $bb_i \equiv \alpha^w s \pmod{p}$.

In the case when $h_i = 1$, the verifier learns whether b is an x_i-blob or a $(1-x_i)$-blob. However, this is of no consequence since he learns nothing about whether b_i is a 0-blob or a 1-blob. It is easy to see that if b were an EPR-blob, the prover would be caught cheating in the above subprotocol with probability $1-2^{-k}$.

We are now ready for the final solution.

3. Sketch of the protocol

Here is a sketch of the promised bounded-round perfect zero-knowledge argumentative protocol for satisfiability. This protocol builds upon the **un**bounded-round perfect zero-knowledge protocol for satisfiability given in [BCC]. The x_i's committed to by the prover in step 2 below are ‘‘scrambled Boolean circuits’’ à la [BCC]. The prover's commitments to the x_i's is realized at the rate of one blob for each bit. To understand the protocol below, it suffices to know that these x_i's are designed in such a way that they can be subjected to two different challenges. In one case, the prover has to open all the blobs, in effect revealing x_i. In the other case, the prover has to selectively open some of the blobs, in effect showing the existence of a satisfying assignment. The only way the prover can meet both challenges is by knowing a satisfying assignment (assuming she is incapable of cheating her bit commitments), but meeting either one of them yields no information to the verifier. For more detail, please consult [BCC].

0) The prover and the verifier agree on p and α and on a safety parameter k (the factorization of $p-1$ is known). This does not count towards the number of rounds in the protocol because it can be done ahead of time and because the numbers p and α could be in the public domain (chosen once and for all by an authority that needs not be trusted).

1) The verifier selects s in $\mathbb{Z}_p^*$, $y_1, y_2, \ldots, y_k$ in $\{0, 1\}$,
   and $r_1, r_2, \ldots, r_k$, $0 \le r_i \le p-2$;
   he computes $c_i = \alpha^{r_i} s^{y_i} \mathbf{mod}\ p$, $1 \le i \le k$;
   he sends $s, c_1, c_2, \ldots, c_k$ to the prover.

2) The prover creates k scrambled circuits $x_1, x_2, \ldots, x_k$ [BCC] and commits to them by sending the verifier one blob for each bit of the x_i's (these blobs are called the *actual* blobs); for each actual blob, the prover also creates k *control* blobs, which she sends to the verifier.

3) For each control blob received by the verifier in step 2, the verifier selects a random challenge h; The verifier sends all of these challenges to the prover.

4) For each challenge h corresponding to actual blob b and control blob $\hat{b}$, the prover either opens blob $\hat{b}$ (if $h = 0$), or shows that she could open blob b if and only if she could open blob $\hat{b}$ (if $h = 1$), as explained at the end of the previous section. (At this point, the verifier is convinced that none of the actual blobs received in step 2 were EPR-blobs.)

5) The verifier opens the blobs $c_1, c_2, \ldots, c_k$ (from step 1) by giving $r_1, r_2, \ldots, r_k$ to the prover. This yields challenges $y_1, y_2, \ldots, y_k$. (The verifier can ''cheat'' here and ''open'' $c_1, c_2, \ldots, c_k$ to show different bits if he knows the discrete logarithm of s, but this is of no consequence as argued previously and proved formally in the next section.)

6) The prover meets the k challenges exactly as in [BCC]: she opens all the actual blobs to show x_i if $y_i = 0$, and she selectively opens some of the blobs for x_i if $y_i = 1$.

4. Proof of the protocol

Two claims must be established: the prover can follow this protocol to the verifier's satisfaction if and only if she knows the secret she claims to have, and the verifier learns nothing about the prover's secret from running the protocol. The first (resp. second) claim must hold even if the prover (resp. verifier) deviates arbitrarily from her (resp. his) prescribed behaviour.

We leave to the reader the details of the proof that the prover cannot succeed in this protocol unless

- she genuinely knows the secret she claims to have; or
- she can compute the discrete logarithm of s while the protocol is in progress (without any help from the verifier); or

- she is very lucky (with probability 2^{-k}, she could guess the k challenges $y_1, y_2, \ldots, y_k$ before she has to commit to her scrambled circuits $x_1, x_2, \ldots, x_k$; also with probability 2^{-k}, she could succeed at creating an EPR-blob that would pass detection at steps 3 and 4).

Therefore, under the assumption that computing discrete logarithms is infeasible (even when the factorization of $p-1$ is known), the prover can cheat only with negligible probability. Furthermore, such attempted cheating would be of the ‘‘daring’’ kind [BCC].

The proof that the protocol is perfect zero-knowledge is more interesting. In order to show that the verifier cannot learn anything even if he deviates arbitrarily from his prescribed behaviour, we must show how to simulate exactly his conversation with the prover in expected polynomial time, without in reality ever talking to the prover. The reader not familiar with the concept of simulation (in the context of zero-knowledge) is referred to [GMR, GMW].

The simulation is made more difficult by the fact that it must be able to handle an honest verifier that follows the protocol when choosing his challenges in step 1 and opening them faithfully at step 5, just as well as a cheating verifier that initially chooses an s for which he knows the discrete logarithm so that he is free to postpone choosing his challenges until step 5. The simulator must also handle strange situations in-between. As always, the simulator will use the verifier as a deterministic ‘‘black box’’ supplied by an external source of randomness under the control of the simulator. By ‘‘the simulator resets the verifier’’, we mean that the simulator sets the verifier back in his starting state and that the simulator will supply the verifier with the same sequence of ‘‘random’’ bits (hence the simulator must store the random bits provided to the verifier, so that she can repeat them after resetting).

The simulation proceeds as follows:

0) The simulator and the verifier agree on p, α and k as in the real protocol. If the verifier refuses to behave properly, the simulator stops.

1) The simulator waits for the verifier to provide s and $c_1, c_2, \ldots, c_k$. If the verifier refuses to behave properly, the simulator stops.

2) The simulator creates as many blobs as the real prover would have to commit to circuits $x_1, x_2, \ldots, x_k$, except that each and every of these blobs is a 0-blob. (This cannot be detected by the verifier since 0-blobs are information-theoretically indistinguishable from 1-blobs.) For each of these blobs, the simulator also provides k control blobs as the real prover would have.

3) The simulator waits for the set of verifier challenges about the control blobs. If the verifier refuses to behave properly, the simulator stops.

4) The simulator meets all the challenges concerning the control blobs, which she can do easily since none of her actual blobs were EPR.

5) The simulator waits for the verifier to open his commitments from step 1, yielding challenge bits $y_1, y_2, \ldots, y_k$ (as well as $r_1, r_2, \ldots, r_k$). If the verifier refuses to behave properly, the simulator stops. (Here, ‘‘improper behaviour’’ includes giving some y_i and r_i such that $c_i \neq \alpha^{r_i} s^{y_i} \textbf{ mod } p$.)

6) At this turning point in the simulation, the simulator resets the verifier and goes over steps 0 and 1 again. It is no longer possible for the verifier to behave improperly since it had not done so the first time around in the same context, with the same random bits, and with the same interaction with the simulator. For the same reason, the verifier will submit the same s and the same c_i’s as he did at step 1.

7) The simulator creates k scrambled circuits $x_1, x_2, \ldots, x_k$ such that she can meet challenge y_i on circuit x_i for each i, $1 \leq i \leq k$. The simulator commits to these k circuits and she emits the usual control blobs.

8) The simulator waits for the set of verifier challenges about the control blobs. If the verifier refuses to behave properly, *the simulator goes back to step 6.*

9) This step is exactly like step 4.

10) The simulator waits for the verifier to open his commitments from step 1 (in fact from step 6, as far as the verifier ‘‘thinks’’), yielding challenge bits $\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_k$ (as well as $\hat{r}_1, \hat{r}_2, \ldots, \hat{r}_k$). If the verifier refuses to behave properly, the simulator goes back to step 6.

11) There are two possibilities at this point: either the verifier did not change any of his challenges as a result of being given a set of blobs at step 7 different from those previously given at step 2, or he changes at least one of them (which he can do only if he knows the discrete logarithm of s).

    (a) If $y_i = \hat{y}_i$ for all i, the simulator is capable of meeting the challenges $\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_k$ because she prepared her scrambled circuits in step 7 precisely with this goal in mind.

    (b) If there is at least one i such that $y_i \neq \hat{y}_i$, the simulator computes $r_i - \hat{r}_i \textbf{ mod } p-1$ or $\hat{r}_i - r_i \textbf{ mod } p-1$, depending on whether $y_i = 0$ or $y_i = 1$, respectively. The result is clearly the discrete logarithm of s. Armed with this information, the simulator can ‘‘cheat’’ in opening the blobs supplied in step 7 in any way she wants, making it child’s play to meet the challenges $\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_k$.

For each set of random bits supplied to the verifier, this simulation yields a probability distribution on the conversation between the verifier and the simulator that is identical with the probability distribution on the conversation between the verifier and the real prover. Also, the expected time for the simulation is only twice the expected time that a conversation with the real prover would have taken. Neither of these claims is immediate because the verifier can decide to behave improperly at step 8 or 10 depending in strange ways on the blobs received from the simulator at step 7. The key to the argument is to consider the probability p that the simulator will reach step 6. Once step 6 is reached, the probability that the verifier will have to be reset at least once is exactly $1-p$. If p is large, the simulator is likely to proceed to step 11 directly, without having had to reset the verifier more than once; if p is small, the simulator is likely to stop quickly, without ever reaching step 6. Fortunately, the bad case for the simulator is unlikely to happen: it is when she reaches step 6 although p was small (in this case, she will probably have to loop back on step 6 a large number of times). We leave the details to the reader.

5. Generalization

We have shown how to obtain a bounded-round perfect zero-knowledge argument under the specific assumption that it is possible 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 $p-1$. This is but a special case of the assumption that one-way group homomorphisms exist (and can be obtained) [IY].

Definition: A function $h : X \rightarrow Y$ is a *one-way group homomorphism* if $(X, *)$ and $(Y, *)$ are finite groups, if h is a homomorphism ($h(x*z) = h(x)*h(z)$ for all $x, z \in X$), and if

1) it is possible to draw efficiently at random into X with uniform distribution;
2) given any $y \in Y$, it is possible to test efficiently whether or not y is in the image of h;
3) the operations $*$ and the computation of inverses in both groups can be performed efficiently; and
4) the homomorphism h is one-way: given any $x \in X$, $h(x)$ can be computed efficiently, but given the value of $h(x)$ for a randomly chosen $x \in X$, it is computationally infeasible to find a $z \in X$ such that $h(z) = h(x)$, except with negligible probability (of course, the above does not exclude finding $z = x$). □

The suggested candidate of one-way group homomorphism we have used so far is $X = (\{0, 1, \ldots, p-2\}, + \textbf{ mod } p-1)$, $Y = (\{1, 2, \ldots, p-1\}, \times \textbf{ mod } p)$ and $h(x) = \alpha^x \textbf{ mod } p$, where p is a prime and α is a generator of Y. It is because of requirement (2) that α must be a generator.

Given any one-way group homomorphism, the verifier can ‘‘set the blobs’’ by choosing $j \in X$ at random and computing $s = h(j)$. The verifier gives s to the prover. The prover commits to bit b by choosing $r \in X$ at random and computing $h(r)$ if $b = 0$ or $s*(h(r))$ if $b = 1$. It is easy to see that the prover cannot open a blob both ways unless she knows how to compute an inverse of s under h. Conversely, it is information-theoretically impossible for the verifier to infer from a blob which bit it encodes. The protocol given in section 3 of this paper can be generalized directly to these blobs.

Here again, either the verifier chooses s in the way stipulated above, in which case he knows its inverse under h and thus the parallel version of the [BCC] protocol is perfect zero-knowledge, or else he chooses s in a way that he does not know its inverse, in which case his commitments $(c_1, c_2, \ldots, c_k)$ to his future challenges $(y_1, y_2, \ldots, y_k)$ are genuine commitments and thus the parallel version of the [BCC] protocol is not less secure than the sequential version (hence, it is perfect zero-knowledge as well). However, a subtlety occurs with respect to the use of ‘‘control blobs’’, which were intended to prevent the prover from issuing EPR-blobs: they work exactly as before, but only if the group X is abelian. Fortunately, control blobs can be replaced by the notion of ‘‘cryptographic capsules’’ [Be], which works even in the non-abelian case.

Finally, note that there is no need for the homomorphism h to be trap-door, and if it is, the trap-door information must kept secret from the prover.

ACKNOWLEDGEMENTS

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

BIBLIOGRAPHY

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

[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, to appear.

[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., Damgaard, 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’’, submitted to *CRYPTO ’89*, March 1989.

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

[G] Goldreich, O., personal communication.

[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.

[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.

[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.