J. Cryptology (1993) 6: 97–116

Journal of
CRYPTOLOGY
© 1993 International Association for Cryptologic Research

A Perfect Zero-Knowledge Proof System for a Problem Equivalent to the Discrete Logarithm*

Oded Goldreich and Eyal Kushilevitz
Department of Computer Science, Technion,
Haifa 32000, Israel
oded@techsel.bitnet eyalk@techunix.bitnet

Communicated by Ernest F. Brickell

Received 6 December 1989 and revised 13 November 1991

Abstract. An interactive proof system is called *perfect zero-knowledge* if the probability distribution generated by any probabilistic polynomial-time verifier interacting with the prover on input theorem φ, can be generated by another probabilistic polynomial-time machine which only gets φ as input (and interacts with nobody!).

In this paper we present a *perfect* zero-knowledge proof system for a decision problem which is computationally equivalent to the Discrete Logarithm Problem. Doing so we provide additional evidence to the belief that *perfect* zero-knowledge proof systems exist in a nontrivial manner (i.e., for languages not in *BPP*). Our results extend to the logarithm problem in any finite Abelian group.

Key words. Interactive proofs, Perfect zero-knowledge, Discrete logarithm.

1. Introduction

One of the most basic questions in complexity theory is how much knowledge should be revealed in order to convince a polynomial-time verifier of the validity of some theorem. This question was raised by Goldwasser *et al.* [GMR], with special emphasis on the extreme case where nothing but the validity of the theorem is given away in the process of proving the theorem. Such proofs are known as *zero-knowledge* proofs and have been the focus of much attention in recent years. Loosely speaking, whatever can be efficiently computed after participating in a zero-knowledge proof can be efficiently computed when just assuming the validity of the assertion.

* This research was partially supported by the Fund for Basic Research Administered by the Israeli Academy of Sciences and Humanities. An early version of this paper appeared in *Advances in Cryptology —Crypto 88* (*Proceedings*), S. Goldwasser (ed.), pp. 57–70, Lecture Notes in Computer Science, vol. 403, Springer-Verlag, Berlin, 1990.

The definition of zero-knowledge considers two types of probability distributions:

1. A distribution generated by a probabilistic polynomial-time verifier after participating in an interaction with the prover.
2. A distribution generated by a probabilistic polynomial-time machine on the input theorem.

Zero-knowledge means that for each distribution of type 1 there exists a distribution of type 2 such that these two distributions are "essentially equal." The exact definition of zero-knowledge depends on the exact interpretation of the phrase "essentially equal." Two extreme cases are of particular interest:

- *Perfect zero-knowledge.* This notion is derived when interpreting "essentially equal" in the most conservative way; namely, exactly equal.
- *Computational zero-knowledge.* This notion is derived when interpreting "essentially equal" in a very liberal way; namely, requiring that the distribution ensembles are polynomially indistinguishable. Loosely speaking, two distribution ensembles are polynomially indistinguishable if they cannot be told apart by any probabilistic polynomial-time test. For a formal definition see [GM] and [Y].

In this paper we focus on the notion of *perfect* zero-knowledge proof systems and provide additional circumstantial evidence to the nontriviality of this notion. The *validity* of our results does not depend on any assumption (yet their *significance* depends on the assumed intractability of the *Discrete Logarithm Problem.*[1])

1.1. *Known Results*

Assuming the existence of secure commitment schemes,[2] Goldreich *et al.* showed that any language in NP has a computational zero-knowledge proof system [GMW]. Using this result it has been shown that whatever can be proven through an efficient interaction proof, can be proven through such a computational zero-knowledge proof [IY], [BGG$^+$]. Thus, assuming the existence of one-way functions, the question of which languages have computational zero-knowledge proof systems is closed.

Much less is known about perfect zero-knowledge. Clearly, any language in *BPP* has a trivial perfect zero-knowledge proof system (in which the prover is inactive). Several languages believed not to be in *BPP* were shown to have perfect zero-knowledge proof systems. These include Quadratic Residuosity and Quadratic non-Residuosity [GMR], Graph Isomorphism and Graph non-Isomorphism

[1] Clearly, if $IP = BPP$, then all languages in IP have trivial perfect zero-knowledge proof systems. Similarly, if the Discrete Logarithm Problem can be solved in probabilistic polynomial-time, then the corresponding decision problem has a trivial perfect zero-knowledge proof system. In such a case our results will remain valid but lose their possible significance.

[2] Using recent results of [N], [H], and [ILL] the existence of commitment schemes can be reduced to the existence of one-way functions.

[GMW], and membership and nonmembership in a subgroup generated by a given group element [TW].[3]

The complexity of languages which have perfect zero-knowledge interactive proof systems was studied by Fortnow [F] and then by Aiello and Hastad [AH]. They prove that if a language L has a perfect zero-knowledge interactive proof system, then both L and $\bar{L}$ have two-step interactive proof systems. Using [Ba] and [GS], this implies that languages having perfect zero-knowledge proof systems fall quite low in the polynomial-time hierarchy (i.e., as low as $\Pi_2^p \cap \Sigma_2^p$). Using a result of Boppana *et al.* [BHZ], such language can also not be NP-complete, unless the polynomial-time hierarchy collapses to its second level.

Perfect zero-knowledge proofs should not be confused with the perfect zero-knowledge *arguments* presented by Chaum [C] and by Brassard *et al.* [BCC]. The difference between an interactive-proof and an *argument* is that the *argument* is sound only if the verifier believes that the prover is a polynomial-time machine with some auxiliary input (which is fixed before the protocol starts). It should be noted however that the class of languages having perfect zero-knowledge *arguments* does not seem to have the same complexity as the class of languages having perfect zero-knowledge proofs. More precisely, it was shown assuming the intractability of the discrete logarithm, that every language that has an interactive proof system has a perfect zero-knowledge *argument* [BCC]. By a recent result of Shamir [S] this implies that assuming the intractability of the discrete logarithm, every language in *PSPACE* has a perfect zero-knowledge *argument*. On the other hand, as mentioned above, every language having a perfect zero-knowledge *proof* is in $\Pi_2^p \cap \Sigma_2^p$. Therefore, if the class of language having perfect zero-knowledge *arguments* equals the class of languages having perfect zero-knowledge *proofs*, then either the discrete-logarithm can be solved in probabilistic polynomial time, or $PSPACE \subseteq \Pi_2^p \cap \Sigma_2^p$.

The work of Brickell *et al.* [BCDG] should not be confused with the present work. Brickell *et al.* present a *computational* zero-knowledge proof system for a problem equivalent to the discrete logarithm problem, assuming the existence of a bit commitment secure against an all-powerful committer. This protocol is hereafter referred to as *version* 1. Using a bit commitment scheme secure against an all-powerful receiver (instead of a bit commitment secure against an all-powerful committer), *version* 2, which constitutes a perfect zero-knowledge *argument* for the same problem is obtained. The first type of bit commitment can be implemented given any one-way function, whereas the second type requires claw-free pairs of one-way permutations [GK] (and both can be implemented assuming the intractability of the Discrete Logarithm Problem). However, neither version yields (or say anything about) *perfect* zero-knowledge *proofs*. The first version is *computational* zero-knowledge (under some intractability assumption) but is *not perfect* zero-knowledge (even if this assumption does hold); whereas the second version is an *argument* (under some intractability assumption) but is *not* an *interactive proof*

[3] It should be noticed that Tompa and Woll's proof of "possession of the Discrete Logarithm" is in fact a proof of membership in a subgroup generated by a primitive element. So are the proofs given by [CEGP] and [CEG].

system (again even if this assumption does hold). Furthermore, the zero-knowledge (resp. soundness) claim of version 1 (resp. version 2), and not just its significance, depends on an intractability assumption. That is, if the discrete logarithm assumption turns out to be false, then version 1 is not zero-knowledge (resp. version 2 is not sound). In contrast, the protocols presented in this paper are perfect zero-knowledge proofs, and this property is independent of any assumption. Hence, the correctness of our work does not depend on any assumption (and only its significance is lost if the discrete logarithm is uniformly easy).

1.2. *Our Results*

In this paper we present a *perfect* zero-knowledge proof system for a decision problem which is computationally equivalent to the Discrete Logarithm Problem. Doing so, we present a perfect zero-knowledge proof system for a problem which is widely believed to be intractable. Thus, we provide additional evidence to the belief that *perfect* zero-knowledge proof systems exist in a nontrivial manner (i.e., for languages not in *BPP*).

Let p be a prime and let g be a primitive element in the multiplicative group modulo p. The *Discrete Logarithm Problem* (*DLP*) is to find, given integers p, g, and y, an integer x such that $g^x \equiv y \bmod p$. Solving *DLP* is believed to be intractable, in particular when $p - 1$ has large prime factors. The best algorithms known for this problem run in subexponential time ($\exp\{O(\sqrt{\log p \log\log p})\}$), see Odlyzko's survey [Od]. It has been shown that determining whether $x \le (p-1)/2$ is computationally equivalent to finding x, on inputs p, g, and $g^x \bmod p$ [BM].[4] This is the case even if x is guaranteed to lie either in the interval $[1, \varepsilon(n)\cdot p]$ or in the interval $[(p-1)/2 + 1, (p-1)/2 + \varepsilon(n)\cdot p]$, where $n^{-O(1)} < \varepsilon(n) < \frac{1}{2}$ and $n = \log_2 p$. This promise problem is hereby referred to as DLP_ε.

In this paper we present a perfect zero-knowledge proof system for DLP_ε. Using the computational equivalence with *DLP*, we have a perfect zero-knowledge proof system for a problem considered computationally hard. Both our protocol and the computational equivalence of *DLP* and DLP_ε extend to any finite Abelian group, in which the group operation can be implemented in polynomial time and the order of the group is known (or can be efficiently found). (In the case of acyclic groups, it is necessary first to define the problems.)

According to the results of [F], [AH], and [BHZ] (see Section 1.1), languages which have perfect zero-knowledge proof systems are not "too high" in the polynomial hierarchy. Furthermore, although perfect zero-knowledge proof systems were given to languages not known to be in *BPP* (see Section 1.1), we believe that these languages are not among the "hardest" and in particular that they are "easier" than *DLP*:

- The Quadratic Residuosity Problem is not "harder" than factoring: Given a polynomial-time algorithm for factoring, we can construct a polynomial-time

[4] In fact, Blum and Micali proved a much stronger statement. Namely, that guessing this bit with success probability greater than $\frac{1}{2} + \varepsilon$ is as hard as retrieving x [BM].

algorithm to decide whether x is a quadratic residue modulo a comosite y. This algorithm will work by first factoring y, and then check whether x is a quadratic residue modulo each prime factor of y (this test can be conducted in polynomial time). On the other hand, it is not known if quadratic residuosity is as "hard" as factoring. It seems plausible that factoring is not "harder" than *DLP* and hence we believe that the Quadratic Residuosity Problem is "easier" than *DLP*.

- The Graph Isomorphism Problem seems to be "easy" for most inputs: For most pairs of graphs (G_1, G_2) it can easily be proved that there is no isomorphism between the two graphs. In fact, this problem is known to be polynomial time on the average for many natural distributions of inputs. For example, in the case that each pair of graphs with the same number of vertices has the same probability [BK] or even in the cae that only d-regular graphs are considered and each pair of d-regular graphs with the same number of vertices has the same probability [Kuc].
- The Subgroup Membership Problem is not "harder" than *DLP*: Given a polynomial-time algorithm for solving *DLP* we can construct a polynomial-time algorithm for determining membership in a subgroup (see Appendix A). In some cases, for example when $p - 1 = 2q$ and q is prime, determining membership in a subgroup of Z_p^* (the multiplicative group mod p) is "easy" (see Appendix B), while solving *DLP* in Z_p^* is considered "hard" also in this case.

2. Preliminaries

2.1. *Promise Problems and Interactive Proofs*

Informally, a *promise problem* is a partial decision problem. That is, a decision problem in which only a subset of all possible inputs is being considered.

Formally a *promise problem* is a pair of predicates (P, Q), where P is called the "promise" and Q is called the "question." A Turing machine M *solves* the promise problem (P, Q) if for every z which satisfies $P(z)$ machine M halts, and it answers "yes" iff $Q(z)$. When $\neg P(z)$ we do not care what M does. This definition originates from [ESY].

We extend to definition of interactive proof systems given by Goldwasser *et al.* [GMR] to promise problems. Intuitively, an interactive proof system for a promise problem (P, Q) is a two-party protocol for a "powerful" *prover P* and a probabilistic polynomial-time *verifier V* satisfying the following two conditions with respect to the common input, denoted z. If $P(z) \wedge Q(z)$, then with a very high probability the verifier is "convinced" of $Q(z)$, when interacting with the prover. If $P(z) \wedge \neg Q(z)$, then no matter what the prover does, he cannot fool the verifier (into believing that "$Q(z)$ is true"), except with a very low probability. When $\neg P(z)$ nothing is required.

Definition 1. An *interactive proof system for a promise problem* (P, Q) is a pair of interacting Turing machines $\langle P, V \rangle$, satisfying the following three conditions:

(0) V is a (probabilistic) expected polynomial-time machine which shares its input with P and they can communicate with each other using special communication tapes.

(1) *Completeness condition*: for every constant $c > 0$, and all sufficiently long z if $P(z) \wedge Q(z)$, then

$$Prob(V \text{ will accept } z \text{ after interacting with } P) \geq 1 - |z|^{-c}.$$

(2) *Soundness condition*: for every Turing machine P^*, every constant $c > 0$, and all sufficiently long z if $P(z) \wedge \neg Q(z)$, then

$$Prob(V \text{ will reject } z \text{ after interacting with } P^*) \geq 1 - |z|^{-c}.$$

2.2. *Perfect Zero-Knowledge Proofs for Promise Problems*

Here, we extend the definition of *perfect zero-knowledge* given by Goldreich *et al.* [GMW] to promise problems.

Definition 2. Let $\langle P, V \rangle$ be an interactive proof system for a promise problem (P, Q), and let V^* be an arbitrary verifier. Denote by $\langle P, V^* \rangle(z)$ the probability distribution on all the read-only tapes of V^* (including the random tape) when interacting with P (the prover) on common input z. We say that the proof system $\langle P, V \rangle$ is *perfect-zero-knowledge* for (P, Q) if, for every expected polynomial-time verifier V^*, there exists a (probabilistic) expected polynomial-time machine, M_{V^*}, such that for every z satisfying $P(z) \wedge Q(z)$ the distributions $M_{V^*}(z)$ and $\langle P, V^* \rangle(z)$ are equal.

2.3. *The Discrete Logarithm Problem and a Related Promise Problem*

Let p be a prime. The set of integers $[1, p-1]$ forms a cyclic group of $p - 1$ elements under multiplication (mod p). This group is denoted Z_p^*. The Discrete Logarithm Problem (DLP) is as follows:

Input: a prime p, a generator $g \in Z_p^*$, and a number $y \in Z_p^*$.
Find $x \in [1, p-1]$ such that $y \equiv g^x \bmod p$. (We use the notation $x = D\log_g y$).

Let g be a primitive element (a generator) of Z_p^* and let y be an element of the group. We define the Half predicate H as follows:

$$H(p, g, y) \quad \Leftrightarrow \quad D\log_g y \in \left[\frac{p-1}{2} + 1, p - 1\right].$$

Let $n = \log_2 p$ and let $\varepsilon(n) < \frac{1}{2}$ be a fraction bounded below by $1/n^{O(1)}$. We define the following predicate:

$$S_\varepsilon(p, g, y) \quad \Leftrightarrow \quad g \text{ is a generator of } Z_p^* \text{ and } D\log_g y \text{ in } [1, \varepsilon(n)(p-1)]$$
$$\text{or} \left[\frac{p-1}{2} + 1, \frac{p-1}{2} + \varepsilon(n)(p-1)\right].$$

When it is clear from the context we shorten $H(p, g, y)$ and $S_\varepsilon(p, g, y)$ to $H(y)$ and $S(y)$, respectively.

The promise problem defined by the pair of predicates (S_ε, H) is hereafter referred to as DLP_ε. Blum and Micali have shown that the DLP_ε is polynomially equivalent to the original DLP in the group Z_p^* [BM].

2.4. *Notation*

- Let s and t be two integers such that $0 \le s \le p-2$ and $0 \le t \le p-2$. $[s, t]$ denotes the set of integers $\{s, s+1, \ldots, t-1, t\}$ in case $s \le t$ or $\{s, s+1, \ldots, p-2, 0, 1, \ldots, t\}$ in case $s > t$. If $s \ge p-1$ or $t \ge p-1$ we take $s \bmod (p-1)$ and $t \bmod (p-1)$ (respectively).
- Let S be a set. The notation $r \in_R S$ means that r is chosen at random with uniform probability distribution among the elements of S.

3. The Protocol for DLP_ε in Z_p^*

In this section we introduce a perfect zero-knowledge protocol for the promise problem DLP_ε. In order to make the protocol more clear we first introduce a protocol which is perfect zero-knowledge only with respect to the honest verifier.

3.1. *Protocol 1—Perfect Zero-Knowledge Proof System with Respect to the Honest Verifier*

Here is a protocol for the promise problem $(S_\varepsilon(p, g, y), H(p, g, y))$ where ε is any constant such that $0 < \varepsilon \le \frac{1}{6}$:

Common input: the integers p, g, and y as previously defined.

The following three steps are executed $n = \log_2 p$ times (unless the verifier *rejects* previously), each time using independent random coin tosses.

(V1) The verifier chooses at random a bit $b \in_R \{0, 1\}$ and an integer $r \in_R [1, 2\varepsilon(p-1)]$. The verifier computes $\alpha = y^b \cdot g^r$ and sends α to the prover.
(P1) The prover computes $\beta = H(\alpha)$ and sends it to the verifier.
(V2) If $\beta \ne b$, then the verifier *rejects*.

If the verifier has completed all n rounds without rejecting, then the verifier *accepts*.

Theorem 1. *Assuming $\varepsilon \le \frac{1}{6}$, then Protocol 1 constitutes an interactive proof system for DLP_ε.*

Proof. Recall that x denotes $D\log_g y$ (i.e., $y \equiv g^x \bmod p$).

Completeness: If $S(y) \wedge H(y)$, then $x \in [(p-1)/2 + 1, (p-1)/2 + \varepsilon(p-1)]$. In addition, V chooses $r \in [1, 2\varepsilon(p-1)]$ where $0 < \varepsilon \le \frac{1}{6}$. Therefore if $b = 0$, then

$$bx + r = r \in [1, 2\varepsilon(p-1)] \subseteq \left[1, \frac{p-1}{2}\right]$$

and if $b = 1$, then

$$bx + r = x + r \in \left[\frac{p-1}{2} + 2, \frac{p-1}{2} + 3\varepsilon(p-1)\right] \subseteq \left[\frac{p-1}{2} + 1, p-1\right]$$

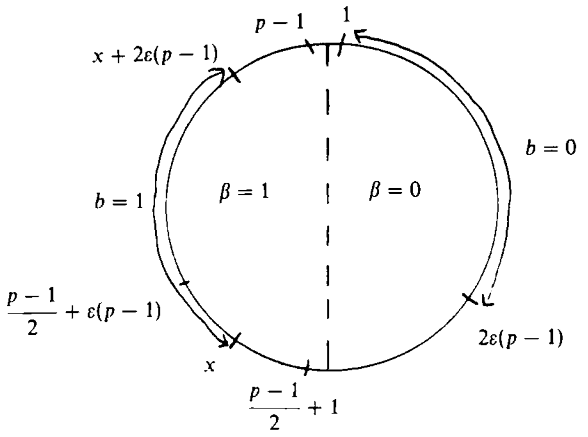

Fig. 1. The completeness condition: $D\log_g \alpha$ as a function of b for inputs with $S(y) \wedge H(y)$.

(see Fig. 1). Therefore in each round

$$\beta = H(\alpha) = H(y^b \cdot g^r) = H(g^{bx+r}) = b.$$

Which means that the prover in this case always causes the verifier to accept the input.

Soundness: If $S(y) \wedge \neg H(y)$, then we have $x \in [1, \varepsilon(p-1)]$. Therefore if the verifier chooses $b = 0$, then $D\log_g \alpha \in [1, 2\varepsilon(p-1)]$, and if it chooses $b = 1$, then $D\log_g \alpha \in [x+1, x+2\varepsilon(p-1)]$ (see Fig. 2). In this case, for any prover P^* we are looking for the probability that V does not reject in a single round (the probabilities are taken over the random choices of V, namely b and r):

$$\begin{aligned} Prob(V \text{ does not reject}) &= Prob(P^*(\alpha) = b) \\ &= Prob(b=0)\cdot Prob(P^*(\alpha) = b \mid b=0) + Prob(b=1) \\ &\quad \cdot Prob(P^*(\alpha) = b \mid b=1) \end{aligned}$$

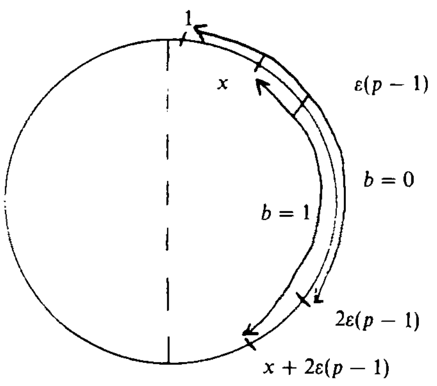

Fig. 2. The soundness condition: $D\log_g \alpha$ as a function of b for inputs with $S(y) \wedge \neg H(y)$.

$$
\begin{aligned}
&= \tfrac{1}{2}\cdot Prob(P^*(g^r) = 0) + \tfrac{1}{2}\cdot Prob(P^*(yg^r) = 1)\\
&= \frac{1}{2}\cdot\frac{1}{2\varepsilon(p-1)}\left(\sum_{i=1}^{x} Prob(P^*(g^i) = 0)\right.\\
&\qquad + \sum_{i=x+1}^{2\varepsilon(p-1)} (Prob(P^*(g^i) = 0) + Prob(P^*(g^i) = 1))\\
&\qquad \left. + \sum_{i=2\varepsilon(p-1)+1}^{2\varepsilon(p-1)+x} Prob(P^*(g^i) = 1)\right)\\
&\le \frac{1}{2}\cdot\frac{1}{2\varepsilon(p-1)}\cdot(x + (2\varepsilon(p-1) - x) + x)\\
&= \frac{1}{2}\cdot\left(\frac{x}{2\varepsilon(p-1)} + 1\right)\\
&\le \frac{1}{2}\cdot\left(\frac{\varepsilon(p-1)}{2\varepsilon(p-1)} + 1\right)\\
&= \tfrac{3}{4}.
\end{aligned}
$$

Therefore in n iterations the probability that the verifier will not reject this input is exponentially low (i.e., $(\frac{3}{4})^n$). □

It is clear that the protocol is perfect zero-knowledge with respect to the honest verifier V. The simulator M_V chooses the random tape for V, and outputs pairs of the form $(\alpha = y^b \cdot g^r, b)$. Since for every y with $S(y) \wedge H(y)$ we have $H(\alpha) = b$ (see the completeness proof) the distribution of M_V's output equals the distribution of conversations between P and the honest verifier V. However, the protocol is probably not zero-knowledge with respect to arbitrary verifiers: a cheating verifier interacting with the prover may send α's for which it wants to know $H(\alpha)$. It could also choose $r \notin [1, 2\varepsilon(p-1)]$ and get in this way some additional information about x. The way to prevent this, is to let the verifier first "prove" to the prover that it "knows" $H(\alpha)$. This is done in the following protocol.

3.2. *Protocol 2—Perfect Zero-Knowledge Proof System with Respect to Any Verifier*

The previous protocol is modified. The modification folows an idea of [GMR] used also in [GMW] and simplified in [Be] (this idea in a different context first appeared in [EGL]). However, in our case the implementation of this idea is more complex.

In the following protocol we provide an interactive proof system to the promise problem $(S_{\varepsilon(n)}(p, g, y), H(p, g, y))$, where $\varepsilon(n)$ will be determined later.

Common input: the integers p, g, and y as previously defined.

The following five steps are repeated $n = \log_2 p$ times (unless the verifier *rejects* or the prover *stops* previously), each time using independent random coin tosses.

(V1) The verifier chooses at random a bit $b \in_R \{0, 1\}$ and an integer

$$r \in_R \left[\left\lfloor \frac{p-1}{4} \right\rfloor + 1, \left\lfloor \frac{p-1}{4} \right\rfloor + \left\lfloor \frac{p-1}{12} \right\rfloor\right].$$

The verifier computes $\alpha = y^b g^r$ and sends α to the prover. In addition to α it computes n pairs of integers. The *ith* pair is denoted α_i and is constructed in the following way: The verifier chooses at random $\pi_i \in_R \{0, 1\}$ and $r_{i,0}$, $r_{i,1} \in_R [1, \lfloor (p-1)/12 \rfloor]$. He computes $\alpha_{i,0} = y^{\pi_i} \cdot g^{r_{i,\pi_i}}$ and $\alpha_{i,1} = y^{\pi_i \oplus 1} \cdot g^{r_{i,\pi_i \oplus 1}}$ (where $\oplus$ denotes the exclusive-or function) and at last sets $\alpha_i = (\alpha_{i,0}, \alpha_{i,1})$. The verifier sends the list of pairs to the prover. Intuitively, every α_i is a random permutation of two elements. Assuming that $S(y) \wedge H(y)$, then one of the elements satisfies $H(\cdot) = 0$ and the other satisfies $H(\cdot) = 1$. These pairs will be used by the verifier to prove that it "knows" $H(\alpha)$.

(P1) The prover chooses at random, a subset $I \subseteq \{1, 2, \ldots, n\}$ with uniform probability distribution among all 2^n subsets. The prover sends I to the verifier.

(V2) If I is not a subset of $\{1, 2, \ldots, n\}$, then the verifier halts and *rejects*. Otherwise, the verifier replies with $\{(\pi_i, r_{i,0}, r_{i,1}): i \in I\}$ and $\{(\pi'_i = \pi_i \oplus b \oplus 1, r'_i = r + r_{i,b \oplus 1}): i \in \bar{I}\}$ (where $\bar{I} = \{1, 2, \ldots, n\} \backslash I$). Intuitively, in this step the verifier proves for every $i \in I$ that the pair α_i is constructed according to the protocol (step V1). In addition, for every $i \in \bar{I}$ the verifier proves (assuming $S(y) \wedge H(y)$) that if α_i is constructed according to step (V1), then it "knows" $H(\alpha)$.

(P2) For every $i \in I$ the prover checks that α_i is constructed according to the protocol (i.e., $r_{i,0}, r_{i,1} \in [1, \lfloor (p-1)/12 \rfloor]$ and $\alpha_i = (y^{\pi_i} \cdot g^{r_{i,\pi_i}}, y^{\pi_i \oplus 1} \cdot g^{r_{i,\pi_i \oplus 1}})$). The prover also checks for every $i \in \bar{I}$ that

$$r'_i \in \left[\left\lfloor \frac{p-1}{4} \right\rfloor + 2, \left\lfloor \frac{p-1}{4} \right\rfloor + 2\left\lfloor \frac{p-1}{12} \right\rfloor\right]$$

and $y \cdot g^{r'_i} = \alpha \cdot \alpha_{i,\pi'_i}$. If either conditions is violated the prover stops. Otherwise, the prover computes $\beta = H(\alpha)$ and sends it to the verifier.

(V3) If $\beta \neq b$, then the verifier *rejects*. Otherwise it continues.

If the verifier has completed all n rounds without rejecting, then the verifier *accepts*.

Theorem 2 (Main Theorem). *Let $\varepsilon(n)$ be a function such that $1/n^{O(1)} < \varepsilon(n) < 1/(100 \cdot n)$, then protocol 2 constitutes a perfect zero-knowledge interactive proof system for DLP_ε.*

Proof. We first prove that Protocol 2 is an interactive proof system for DLP_ε, and then we show that it is perfect zero-knowledge. Recall again that $x = D \log_g y$.

Completeness: Similar to the completeness in Theorem 1.

Soundness: We prove that although in some cases α and the list of pairs $S = \{\alpha_1, \ldots, \alpha_n\}$ can yield information about b, there is a big enough probability that α and S will not yield any information about b, and therefore will not help the prover to convince V that

$$x \in \left[\frac{p-1}{2} + 1, \frac{p-1}{2} + \varepsilon(n)\cdot(p-1)\right]$$

when in fact $x \in [1, \varepsilon(n)\cdot(p-1)]$.

Assume that $x \in [1, \varepsilon(n)\cdot(p-1)]$. We call α *good* if it satisfies

$$D \log_g \alpha \in \left[\left\lfloor\frac{p-1}{4}\right\rfloor + x + 1, \left\lfloor\frac{p-1}{4}\right\rfloor + \left\lfloor\frac{p-1}{12}\right\rfloor\right].$$

Otherwise α is *bad*. Intuitively, when α is good the prover cannot learn anything about b from α (since in this case $Prob(b = 0 | y^b \cdot g^r = \alpha) = \frac{1}{2}$). The probability that α is bad is less than or equal to $12\cdot\varepsilon(n)$.

Similarly we call a pair α_i *good* if both $r_{i,0}$ and $r_{i,1}$ are in $[x + 1, \lfloor(p-1)/12\rfloor]$. Otherwise α_i is *bad*. The list of pairs S is *good* if every α_i is good, and is *bad* otherwise. The probability that a pair α_i is bad is less than $2\cdot 12\cdot\varepsilon(n)$ and the probability that S is bad is therefore less than $24n\cdot\varepsilon(n)$.

We remark here that since P^* has infinite power we can assume without loss of generality that P^* is deterministic.[5] Therefore for any α and S the prover P^* always chooses the same subset I, denoted $f(\alpha, S)$.

Our first claim is that in every round in which α and S are *good*, the prover does not get any information about b. Clearly, the answers $\{(\pi_i, r_{i,0}, r_{i,1})\}_{i\in I}$ (which the verifier sends in step (V2)) are independent of b and therefore are irrelevant. However, we claim that also α, S, and the answers for $i \in \bar{I}$ do not help the prover to find what b is. Formally, for every *good* α and S, and for every $\{r'_i, \pi'_i\}_{i\in\bar{I}}$,

$$Prob(b = 0 | y^b g^r = \alpha \wedge f(\alpha, S) = I \wedge \forall i \in \bar{I}(y\cdot g^{r'_i} = \alpha\cdot\alpha_{i,\pi'_i} \wedge r'_i = r + r_{i,\pi'_i})) = \tfrac{1}{2}.$$

The reason is that when α and S are good then (by the definition of "*good*") assigning any value to b yields unique values to all the other variables r, $r_{i,0}$, and $r_{i,1}$. Thus, there are only two elements in the conditional probability space, one corresponds to $b = 0$ and the other to $b = 1$. Using this claim we now show that the probability that P^* will convince V in a single round is "low":

$$\begin{aligned}
&Prob(P^*(\alpha, S, \{\pi_i, r_{i,0}, r_{i,1} : i \in f(\alpha, S)\}, \{\pi'_i, r'_i : i \notin f(\alpha, S)\}) = b)\\
&\le Prob(P^*(\alpha, S, \{\pi_i, r_{i,0}, r_{i,1} : i \in f(\alpha, S)\}, \{\pi'_i, r'_i : i \notin f(\alpha, S)\})\\
&\qquad = b | \alpha \text{ and } S \text{ are good}) + Prob(\alpha \text{ is bad or } S \text{ is bad})\\
&\le \tfrac{1}{2} + (24n + 12)\cdot\varepsilon(n).
\end{aligned}$$

[5] Informally, the reason for this is as follows: suppose there exists a probabilistic prover P^{**} that is able to convince V with probability p_0. Then there must exist "good" coin tosses for the first round of P^{**} that guarantee that if P^{**} uses these coin tosses, then V is convinced with probability $\ge p_0$. As P^* is all-powerful and as V is a *fixed* probabilistic polynomial-time machine, P^* can deterministically find such "good" coin tosses and simulate P^{**} with these coin tosses. The argument for the next rounds is similar.

Since $\varepsilon(n) < 1/(100 \cdot n)$ this probability is less than $\frac{3}{4}$ (as long as $n > 12$). Therefore the probability that P^* will mislead V (i.e., provide correct β's) in all n rounds is exponentially low.

Zero-knowledge: For every expected polynomial-time interactive machine V^*, we present a machine M_{V^*} so that, for every input satisfying $S_{\varepsilon(n)}(p, g, y) \wedge H(p, g, y)$, $M_{V^*}(p, g, y) = \langle P, V^* \rangle (p, g, y)$. The machine M_{V^*} uses V^* as a subroutine.

The idea of the simulator M_{V^*} is to cause V^* to yield all the information needed for calculating $H(\alpha)$. This is done by executing V^* several times with the same random tape, so that V^* will send the same α and S. Machine M_{V^*} will try to get, for one of the pairs α_i, the information $\{\pi_i, r_{i,0}, r_{i,1}\}$ in one trial and $\{\pi'_i, r'_i\}$ in another. If this information is constructed according to the protocol (M_{V^*} will check it), then this is enough for calculating $H(\alpha)$.

Following is a detailed description of M_{V^*}. For simplicity we start by dealing with the case that V^* is polynomial time, and only then do we deal with the case of expected polynomial-time verifiers. The machine M_{V^*} starts by choosing a random tape $s \in_R \{0, 1\}^q$ for V^*, where $q = poly(|p, g, y|)$ is a bound on the running time of V^* on the current input. (Clearly, V^* reads at most q bits from its random tape.) M_{V^*} places s on its record tape and proceeds in n rounds as follows.

Round j:

(S1) Machine M_{V^*} initiates V^* on the input (p, g, and y), the random tape s, and the sequence of previous successfully simulated $j - 1$ communication rounds. It reads from the communication tape of V^* the element α and the pairs $\alpha_1 \cdots \alpha_n$. Machine M_{V^*} chooses a random subset I and places it on the communication tape of V^*. Machine M_{V^*} also appends I to its record tape.

(S2) M_{V^*} reads from the communication tape of V^* $\{(\pi_i, r_{i,0}, r_{i,1}): i \in I\}$ and $\{(\pi'_i, r'_i): i \in \bar{I}\}$. For every $i \in I$ machine M_{V^*} checks whether $\pi_i \in \{0, 1\}$, $r_{i,0}, r_{i,1} \in [1, \lfloor (p-1)/12 \rfloor]$, and whether $\alpha_i = (y^{\pi_i} \cdot g^{r_{i,\pi_i}}, y^{\pi_i \oplus 1} \cdot g^{r_{i,\pi_i \oplus 1}})$. It also checks for every $i \in \bar{I}$ whether

$$r'_i \in \left[\left\lfloor \frac{p-1}{4} \right\rfloor + 2, \left\lfloor \frac{p-1}{4} \right\rfloor + 2 \left\lfloor \frac{p-1}{12} \right\rfloor \right]$$

and $y \cdot g^{r'_i} = \alpha \cdot \alpha_{i,\pi'_i}$. If either condition is violated M_{V^*} outputs its record tape and stops. Otherwise, M_{V^*} continues to step (S3).

(S3) The purpose of this step is to find $H(\alpha)$. This is done by repeating the following procedure (until $H(\alpha)$ is found):

(S3.1) Machine M_{V^*} chooses at random a subset $K \subseteq \{1, 2, \ldots, n\}$ not equal to I. Machine M_{V^*} initiates V^* on the same input, the same (!) random tape s, and the same sequence of previous successfully simulated rounds. It places K on the read-only communication tape of V^*. Subsequently, machine M_{V^*} reads from the communication tape of V^* $\{(\delta_i, s_{i,0}, s_{i,1}): i \in K\}$ and $\{(\delta'_i, s'_i): i \in \bar{K}\}$.

(S3.2) Machine M_{V^*} checks whether the information it received is correct. (The same tests as it does for the answers to I.) If it is not correct, then M_{V^*} goes back to step (S3.1). Otherwise M_{V^*} finds i such that $i \in I \cap \bar{K}$

or $i \in \bar{I} \cap K$. Such an i exists since $I \neq K$, without loss of generality we assume that $i \in I \cap \bar{K}$. (The index i corresponds to a pair α_i for which V^* sent (in a different trials) both $\{\pi_i, r_{i,0}, r_{i,1}\}$ and $\{\pi_i', r_i'\}$. Since all this information was checked by M_{V^*} and found correct then M_{V^*} is now able to compute $H(\alpha)$.) Machine M_{V^*} sets $\beta = \pi_i \oplus \delta_i' \oplus 1$.

(S3.3) In parallel to (S3.1) and (S3.2), try to find $H(\alpha)$ by exhaustive search. (Make one try for each invocation of V^*.)

(S4) Once β is found, machine M_{V^*} appends β to its record tape, thus completing round j.

If all rounds are completed, then M_{V^*} outputs its record tape and halts.

We now have to prove the validity of the construction. First we prove that the simulator M_{V^*} indeed terminates in expected polynomial time. Next, we prove that the output distribution produced by M_{V^*} does equal the distribution over V^*'s tapes (when interacting with P). Once these two claims are proven, the theorem follows.

Claim 1. *Machine M_{V^*} terminates in expected polynomial time.*

Proof. We consider the expected running time on a single round j with respect to a particular random tape s and a *fixed* sequence of communications corresponds to the first $j-1$ rounds. We call a subset $I \subseteq \{1, 2, \ldots, n\}$ *good* if V^* answers properly on message I with random tape s. Denote by g_s the number of good subsets with respect to random tape s. Clearly, $0 \leq g_s \leq 2^n$. We compute the expected number of times V^* is invoked in round j as a function of g_s. We need to consider three cases:

Case 1 ($g_s \geq 2$). In case the subset I chosen in step (S1) is good, we have to consider the probability that another subset K is also good. In case the set I chosen in step (S1) is bad, the round is completed immediately. Thus, the expected number of invocations is

$$\frac{g_s}{2^n} \cdot \left(\left(\frac{g_s - 1}{2^n - 1} \right)^{-1} + 1 \right) + \frac{2^n - g_s}{2^n} \cdot 1 < \frac{g_s}{g_s - 1} + 1 \leq 3.$$

Case 2 ($g_s = 1$). With exponentially small probability (i.e., 2^{-n}) the subset I chosen in step (S1) is good. In this case we find β by exhaustive search (in stage (S3.3)). Otherwise, the round is completed immediately. Thus, the expected complexity of M_{V^*} in Case 2 is bounded by one invocation of V^* and an additional $(p-1) \cdot 2^{-n} \leq 1$ step.

Case 3 ($g_s = 0$). The subset I chosen in step (S1) is always bad, and thus M_{V^*} invokes V^* exactly once and then halts.

The claim follows by additivity of expectation and the fact that V^* is polynomial time. □

Claim 2. *The probability distribution $M_{V^*}(p, g, y)$ is identical to the distribution $\langle P, V^* \rangle (p, g, y)$.*

Proof. Both distributions consists of a random s, and sequence of elements, each being either (I, β) (with good I) or a bad I, with random I. In $\langle P, V^*\rangle(p, g, y)$ we have $\beta = H(\alpha)$ (where $\alpha = V^*(p, g, y, s)$) we need to show that this is the case also in $M_{V^*}(p, g, y)$, i.e., we prove that when I is good then M_{V^*} succeeds in finding $H(\alpha)$, but this is true because either it finds $H(\alpha)$ by exhaustive search or find an i in which π_i, $r_{i,0}$, $r_{i,1}$, δ_i', and s_i' are all correct. That is,

(a) $r_{i,0}, r_{i,1} \in [1, \lfloor(p-1)/12\rfloor]$,
(b) $s_i' \in [\lfloor(p-1)/4\rfloor + 2, \lfloor(p-1)/4\rfloor + 2\lfloor(p-1)/12\rfloor]$,
(c) $\alpha_{i,j} = y^{\pi_i \oplus j} \cdot g^{r_{i,\pi_i \oplus j}}$, and
(d) $y \cdot g^{s_i'} = \alpha \cdot \alpha_{i,\delta_i'}$.

In this case we have (using (d) and then (c))

$$\begin{aligned} H(\alpha) &= H(y \cdot g^{s_i'} \cdot (\alpha_{i,\delta_i'})^{-1}) \\ &= H(y \cdot g^{s_i'} \cdot (y^{\pi_i \oplus \delta_i'} \cdot g^{r_{i,\pi_i \oplus \delta_i'}})^{-1}) \\ &= H(y^{\pi_i \oplus \delta_i' \oplus 1} \cdot g^{s_i' - r_{i,\pi_i \oplus \delta_i'}}). \end{aligned}$$

Finally, using (a) and (b) we have $s_i' - r_{i,\pi_i \oplus \delta_i'} \in [2/12(p-1), 5/12(p-1)]$. In addition, $D \log_g y \in [(p-1)/2 + 1, (p-1)/2 + \varepsilon(n)(p-1)]$, Therefore,

$$H(\alpha) = H(y^{\pi_i \oplus \delta_i' \oplus 1}) = \pi_i \oplus \delta_i' \oplus 1.$$

This is exactly the way that M_{V^*} computes $H(\alpha)$. □

This completes the proof of zero-knowledgeness with respect to polynomial-time verifiers. For the case where V^* is expected polynomial time we need to perform some technical modification of M_{V^*}. This follows from the fact that there may be no polynomial bound on the length of the random tape of V^*. This is solved by letting M_{V^*} select the random tape of V^* adaptively: each time V^* requires a new random bit, M_{V^*} choose it at random and also store it for all future invocations of V^*. Using this modification the proof remains valid also for expected polynomial-time verifiers. □

Remark 1. The protocol obtained by executing n copies of the give steps of Protocol 2 in parallel (instead of executing them sequentially) is also perfect zero-knowledge (for details see [Kus]).

Remark 2. The DLP_ε can be extended to any two intervals of size $\varepsilon(n) \cdot (p-1)$, where the first interval starts from s and the second from $s + (p-1)/2 \bmod p - 1$. This promise problem is hereby referred to as DLP_ε'. The problems DLP_ε and DLP_ε' are clearly polynomially equivalent (see Lemma 1, Appendix C). A zero-knowledge protocol for DLP_ε' on input (p, g, y, s) will work by first computing $y' = y \cdot g^{-s+1} \bmod p$ and then executing the protocol for DLP_ε on (p, g, y').[6]

[6] We remark that the fact that the latter protocol is indeed zero-knowledge is not implied merely by the existence of a polynomial-time reduction from DLP_ε' to DLP_ε, since the verifier while executing the protocol for DLP_ε has additional information about the input of this protocol, which is the original input for the DLP_ε'. However, since the protocol for DLP_ε was proved to be zero-knowledge using a black-box simulation, then it is also auxiliary-input zero-knowledge [Or], [GO]. This implies the zero-knowledgeness of the modified protocol.

4. Extensions

4.1. *Generalization of the Protocol to Other Cyclic Groups*

Let $\{G_i\}$ be a family of cyclic groups such that the following conditions hold:

1. The group operation of every G_i can be implemented in polynomial time. That is, there exists a polynomial-time Algorithm A that on input (i, x, y) outputs $x \cdot y$ for every $x, y \in G_i$.
2. The order of G_i (to be denoted N_i) is either given or can be computed in polynomial time (i.e., there exists a polynomial-time Algorithm B that on input i outputs N_i).

We can extend the definitions of the *DLP* and the DLP_ε in the obvious way. The modifications required are to replace any multiplication mod p by the group operation of G_i and to replace $p - 1$ by the group order (N_i) (for example, the *DLP* problem is given i and $g, y \in G_i$, where g is a generator of G_i, to find x such that $y = g^x$).

With the same modifications our protocol is a perfect zero-knowledge proof system for the promise problem DLP_ε in $\{G_i\}$ (since the protocol does not make any use of the special structure of Z_p^*, but merely its being cyclic). What we still have to show is that DLP_ε is polynomially equivalent to *DLP* in any such family. The Blum–Micali proof (used for the group Z_p^*) extends easily *only* to groups in which the order is even and both testing quadratic residuosity and taking square root can be performed in polynomial time. Unfortunately, this does not seem to be the case in all groups and a different argument is needed.

Theorem 3. *For every family of cyclic groups $\{G_i\}$ that satisfies the above conditions and for every function $\varepsilon(n)$ such that $n^{-O(1)} < \varepsilon(n) < \frac{1}{2}$ where $n = \log_2 N_i$, the problems DLP and DLP_ε are polynomially equivalent.*

In Appendix C we present a proof for this theorem based on ideas of Kaliski [Ka]. (The proof in [Ka] is more complicated since he proves a stronger statement.)

4.2. *Generalization of the Results to Acyclic Groups*

Let $\{G_i\}$ be a family of acylic groups which are finite and Abelian. In this case G_i does not have a generator but a *generating-tuple* $\bar{g} = (g_1, g_2, \ldots, g_k)$. Any element $y \in G_i$ can be uniquely expressed as $y = g_1^{x_1} \cdots g_k^{x_k}$. The order of each g_j is denoted $N_i(g_j)$ and the number of elements in the group is $N_i = N_i(g_1) \cdot N_i(g_2) \cdots N_i(g_k)$. We also assume that for every group G_i, the group operation, the order N_i, and also $N_i(g_1)$ are polynomial-time computable.

The problems *DLP* and DLP_ε are defined with respect to g_1. (For example the *DLP* in such a family is: given i, y, and $\bar{g}$ where $y, g_1, \ldots, g_k \in G_i$ and $\bar{g}$ is a generating tuple of G_i, find x_1 such that $\exists x_2 \cdots x_k | y = g_1^{x_1} \cdots g_k^{x_k}$). Our protocol with some modifications works here too. We should replace every occurrence of $p - 1$ by $N_i(g_1)$ and also everything done with respect to g has to be done with respect to g_1. In addition we should randomize everything by elements chosen at random from the subgroup generated by $(g_2, g_3, \ldots, g_k)$. For example in step (V1) of the protocol the verifier should compute $\alpha = y^b \cdot g_1^{r_1} \cdot g_2^{r_2} \cdots g_k^{r_k}$, where $b \in_R \{0, 1\}$,

$$r_1 \in_R \left[\left\lfloor \frac{N_i(g_1)}{4} \right\rfloor + 1, \left\lfloor \frac{N_i(g_1)}{4} \right\rfloor + \left\lfloor \frac{N_i(g_1)}{12} \right\rfloor\right],$$

and $r_2 \cdots r_k \in_R [1, N_i]$.

Using the same modifications described above we can also modify Theorem 3 to show that the *DLP* and the DLP_ε are still equivalent in such an acyclic group.

Appendix A. Subgroup Membership is not Harder than Discrete Logarithm

In this appendix we prove that the Subgroup Membership Problem is not "harder" than the Discrete Logarithm Problem. We do so by presenting an expected polynomial-time algorithm that uses an oracle *LOG* for the Discrete Logarithm Problem, and solves the Subgroup Membership Problem. We start by giving a formal definition of the Subgroup Membership Problem:

Input: a prime p, and $a, b \in Z_p^*$.
Find whether a belongs to the subgroup defined by b. That is, whether there exists i such that $a \equiv b^i \bmod p$.

The algorithm on input p, a, and b works as follows (as usual, we denote $n = \log_2 p$):

1. The target of this step is to find g which is a generator of Z_p^*:
   Choose a candidate $g \in_R Z_p^*$.
   Choose n random elements $y_1, y_2, \ldots, y_n \in_R Z_p^*$. For each of them compute:
   $$x_i = LOG(p, g, y_i).$$
   Verify for every i that $y_i \equiv g^{x_i} \bmod p$. If not, then find another candiate, otherwise go to step (2).
2. The target of this step is to check (using g that we found in step 1) whether a belongs to the subgroup defined by b. We compute:
   $$\alpha = LOG(p, g, a),$$
   $$\beta = LOG(p, g, b).$$
   If α does not saitsfy $a \equiv g^\alpha \bmod p$ or β does not satisfy $b \equiv g^\beta \bmod p$, then return to step 1. (This can happen only in the case where g is not a generator.) If α and β are correct, then compute $t = \gcd(\beta, p - 1)$. If t divides α answer **YES** otherwise answer **NO**.

Since at least $\Omega(1/\log n)$ of the group elements are generators (see [RS]), the expected number of times we will have to execute step 1 in order to find a generator is $O(\log n)$. In the case that g is not a generator, the probability that it will pass all the n tests in step 1 is at most 2^{-n} (since its order is at most half of the group order). Finally, it can be verified that if g is indeed a generator, then $\gcd(\beta, p - 1)$ divides α if and only if there exists i such that $a \equiv b^i \bmod p$ (note: it always holds that $\gcd(\beta, p - 1)$ divides α if and only if $\exists i$; $\alpha = \beta \cdot i \bmod (p - 1)$). Assuming that *LOG* is polynomial time then it follows that the above algorithm is correct and runs in expected polynomial time.

Appendix B. Determining Membership in a Subgroup—Special Case

In this appendix we consider the problem of determining membership in a subgroup generated by an element g in Z_p^*, when p is a prime satisfying $p - 1 = 2q$ for some prime q. We show that in this special case, testing membership in a subgroup is easy. This should be contrasted with the believed intractability of *DLP* also for this case.

It can be readily verified that if $p - 1 = 2q$ with q prime, then Z_p^* has $q - 1$ primitive elements (i.e., elements of order $p - 1$), $q - 1$ elements of order q, one element of order 2, and one element of order 1 (i.e., the identity). Furthermore, all the elements of order q and the identity element form a subgroup which is generated by any of the elements of order q. Thus, the question of whether a is in the subgroup generated by b reduces (in this case!) to testing the order of both a and b (a is in the subgroup generated by b iff the order of a divides the order of b). Finally note that testing the order of an element is easy (in this case!).

Appendix C. The Equivalence of *DLP* and DLP_ε

In this appendix we prove Theorem 3 (Section 4.1), claiming the equivalence of *DLP* and DLP_ε for every family of cyclic groups $\{G_i\}$ in which the group operation and the group order are polynomial-time computable. For proving this theorem we first prove the following two lemmas.

Lemma 1. *For every family of cyclic groups* $\{G_i\}$ *as above, and for every function* $\varepsilon(n)$ *such that* $n^{-O(1)} < \varepsilon(n) < \frac{1}{2}$ *(where* $n = \log_2 N_i$*) the problems* DLP_ε *and* DLP'_ε *are polynomially equivalent.*

Proof. We show a polynomial-time reduction from each problem to the other. Given $w = (i, g, y)$, an input for the DLP_ε problem, we transform it to $w' = (i, g, y, 1)$. Given $w' = (i, g, y, s)$, an input for the DLP'_ε problem, we transform it to $w = (i, g, y \cdot g^{-s+1})$. In both cases it can be easily verified that $w \in DLP_\varepsilon$ if and only if $w' \in DLP'_\varepsilon$. □

Lemma 2. *For every cyclic group G of order N, for every* $y \in G$, *and for every* $d < N$: *if* $D \log_g y^2 \in [s, s + d]$, *then* $D \log_g y$ *is in* $[\lceil s/2 \rceil, \lceil s/2 \rceil + \lceil d/2 \rceil]$ *or in* $[\lceil (s + N)/2 \rceil, \lceil (s + N)/2 \rceil + \lceil d/2 \rceil]$.

Proof. Let $w \in [s, s + d] \subseteq \{0, 1, \ldots, N - 1\}$. We show that every solution of the equation $2 \cdot x \equiv w \bmod N$ satisfies $x \in [\lceil s/2 \rceil, \lceil s/2 \rceil + \lceil d/2 \rceil]$ or $x \in [\lceil (s + N)/2 \rceil, \lceil (s + N)/2 \rceil + \lceil d/2 \rceil]$. We deal with two cases:

Case 1: *N is odd.* Since N is odd, 2 has an inverse module N which is $(N + 1)/2$. In this case if w is even, then $x = w/2 \in [\lceil s/2 \rceil, \lceil s/2 \rceil + \lceil d/2 \rceil]$ and if w is odd, then

$$x = \frac{w + N}{2} \in \left[\left\lceil \frac{s + N}{2} \right\rceil, \left\lceil \frac{s + N}{2} \right\rceil + \left\lceil \frac{d}{2} \right\rceil\right].$$

Case 2: *N is even.* In this case, 2 does not have an inverse modulo N, and therefore the above equation has a solution only when w is even. In such a case the equation has two solutions: $x_1 = w/2$ and $x_2 = (w + N)/2$. These solutions satisfy

$$x_1 = \frac{w}{2} \in \left[\left\lceil \frac{s}{2} \right\rceil, \left\lceil \frac{s}{2} \right\rceil + \left\lceil \frac{d}{2} \right\rceil\right]$$

and

$$x_2 = \frac{w+N}{2} \in \left[\left\lceil \frac{s+N}{2} \right\rceil, \left\lceil \frac{s+N}{2} \right\rceil + \left\lceil \frac{d}{2} \right\rceil\right].$$

Now taking $w = D\log_g y^2$ and $x = D\log_g y$, the lemma follows. □

Proof of Theorem 3. By Lemma 1 it is enough to prove the equivalence of DLP and DLP'_ε. It is obvious that if we know how to solve the DLP we can solve the DLP'_ε. We prove the other direction by presenting an algorithm that solves the DLP using an oracle $HALF_G(g, y, s)$ that solves the DLP'_ε. The oracle $HALF_G$ is defined as follows:

$$HALF_G(g, y, s) = \begin{cases} 0, & D\log_g y \in [s, s + \varepsilon(n)\cdot N], \\ 1, & D\log_g y \in [s + \lfloor N/2 \rfloor, s + \lfloor N/2 \rfloor + \varepsilon(n)\cdot N], \\ ?, & \text{otherwise.} \end{cases}$$

where $1/n^{O(1)} < \varepsilon(n) < \frac{1}{2}$ and $n = \log_2 N$. By "?" we mean that in this case $HALF_G$ can give any output. The following algorithm solves the DLP using this oracle:

Algorithm 1. (The input is $y \in G$ and a generator g.)

```
1. Let $n = \log_2 N$.
2. Compute $y_1 \leftarrow y, y_2 \leftarrow y_1^2, y_3 \leftarrow y_2^2, \ldots, y_n \leftarrow y_{n-1}^2$.   /* $y_i = y^{2^{i-1}}$ */
3. Let $s, s' \leftarrow 0$.
4. For $k = n$ to 1 do
     If ($HALF_G(g, y_k, s) = 0$), then $s \leftarrow s$   /* i.e., do nothing */
       else $s \leftarrow \left(s + \left\lfloor \frac{N}{2} \right\rfloor\right) \bmod N$
     $s \leftarrow \lceil s/2 \rceil$
   end
5. If ($g^s = y$) output $s$
   else if ($g^{s+1} = y$) output $s + 1$
6. $s' \leftarrow s' + \varepsilon(n)\cdot N$
   $s \leftarrow s'$; goto (4)
```

The algorithm finds $D\log_g y$ by looking for $D\log_g y_n = D\log_g y^{2^n}$ in $1/\varepsilon(n) = n^{O(1)}$ disjoint intervals of size $\varepsilon(n)\cdot N$. When we are looking in the correct interval then, according to Lemma 2, we can find $D\log_g y$ using a binary search. Namely, if $D\log_g y_i$ belongs to an interval of size d, then $D\log_g y_{i-1}$ belongs to one of two intervals of size $\lceil d/2 \rceil$. We decide which interval is the correct one by using the oracle

$HALF_G$. Therefore after at most $n = \log_2 N$ rounds we are looking for $D \log_g y_1 = D \log_g y$ in an interval of size 2. Now we check which of the two numbers in the interval is $D \log_g y$. If both do not fit, then the current interval is wrong and we try another one. Since there are polynomially many intervals, and since the cost of the binary search is polynomial (assuming that $HALF_G$ is polynomial time, and that computing the order of G and the group operation are polynomial time), the algorithm is polynomial time. □

References

[AH] Aiello, W., and J. Hastad, Perfect Zero-Knowledge Languages Can Be Recognized in Two Rounds, *Proc. 28th FOCS*, 1987, pp. 439–448.

[Ba] Babai, L., Trading Group Theory for Randomness, *Proc. 17th STOC*, 1985, pp. 421–429.

[BK] Babai, L., and L. Kucera, Canonical Labeling of Graphs in Linear Average Time, *Proc. 20th FOCS*, 1979, pp. 39–46.

[Be] Benaloh (Cohen), J. D., Cryptographic Capsules: A Disjunctive Primitive for Interactive Protocols, *Advances in Cryptology—Crypto 86 (Proceedings)*, A. M. Odlyzko (ed.), pp. 213–222, Lecture Notes in Computer Science, Vol. 263, Springer-Verlag, Berlin, 1987.

[BGG$^+$] Ben-or, M., O. Goldreich, S. Goldwasser, J. Hastad, J. Kilian, S. Micali, and P. Rogaway, Everything Provable Is Provable in Zero-Knowledge, *Advances in Cryptology—Crypto 88 (Proceedings)*, S. Goldwasser (ed.), pp. 37–56, Lecture Notes in Computer Science, Vol. 403, Springer-Verlag, Berlin, 1990.

[BM] Blum, M., and S. Micali, How To Generate Cryptographically Strong Sequences of Pseudo-Random Bits, *SIAM J. Comput.*, Vol. 13, 1984, pp. 850–864.

[BHZ] Boppana, R., J. Hastad, and S. Zachos, Does Co-NP Have Short Interactive Proofs?, *Inform. Process. Lett.*, Vol. 25, May 1987, pp. 127–132.

[BCC] Brassard, G., D. Chaum, and C. Crepeau, Minimum Disclosure Proofs of Knowledge, *J. Comput. System Sci.*, Vol. 37, No. 2, October 1988, pp. 156–189.

[BCDG] Brickell E. F., D. Chaum, I. Damgard, and J. van de Graaf, Gradual and Verifiable Release of a Secret, *Advances in Cryptology—Crypto 87 (Proceedings)*, C. Pomerance (ed.), pp. 156–166, Lecture Notes in Computer Science, Vol. 293, Springer-Verlag, Berlin, 1987.

[C] Chaum, D., Demonstrating that a Public Predicate Can be Satisfied Without Revealing Any Information About How, *Advances in Cryptology—Crypto 86 (Proceedings)*, A. M. Odlyzko (ed.), pp. 195–199, Lecture Notes in Computer Science, Vol. 263, Springer-Verlag, Berlin, 1987.

[CEG] Chaum, D., J. H. Evertse, and J. van de Graaf, An Improved Protocol for Demonstrating Possession of a Discrete Logarithm Without Revealing It, *Advances in Cryptology—Eurocrypt 87 (Proceedings)*, D. Chaum and W. L. Price (eds.), pp. 127–142, Lecture Notes in Computer Science, Vol. 304, Springer-Verlag, Berlin, 1988.

[CEGP] Chaum, D., J. H. Evertse, J. van de Graaf, and R. Peralta, Demonstrating Possession of a Discrete Logarithm Without Revealing It, *Advances in Cryptology—Crypto 86 (Proceedings)*, A. M. Odlyzko (ed.), pp. 200–212, Lecture Notes in Computer Science, Vol. 263, Springer-Verlag, Berlin, 1987.

[EGL] Even, S., O. Goldreich, and A. Lempel, A Randomized Protocol for Signing Contracts, *Comm. ACM*, Vol. 28, No. 6, 1985, pp. 637–647.

[ESY] Even, S., A. L. Selman, and Y. Yacobi, The Complexity of Promise Problems with Applications to Public-Key Cryptography *Inform. Control*, Vol. 61, 1984, pp. 159–173.

[F] Fortnow, L., The Complexity of Perfect Zero-Knowledge, *Proc. 19th STOC*, pp. 204–209, 1987.

[GK] Goldreich, O., and A. Kahn, in preparation.

[GMW] Goldreich, O., S. Micali, and A. Wigderson, Proofs that Yield Nothing but Their Validity and a Methodology of Cryptographic Protocol Design, *J. Assoc. Comput. Math.*, Vol. 38, No. 1, 1991, pp. 691–729.

[GO] Goldreich, O., and Y. Oren, On the Cunning Power of Cheating Verifiers: Some Observations about Zero-Knowledge Proofs, in preparation.

[GM] Goldwasser, S., and S. Micali, Probabilistic Encryption, *J. Comput. System Sci.*, Vol. 28, No. 2, 1984, pp. 270–299.

[GMR] Goldwasser, S., S. Micali, and C. Rackoff, The Knowledge Complexity of Interactive Proof Systems, *SIAM J. Comput.*, Vol. 18, No. 1, 1989, pp. 186–208. Early version appeared in *Proc. 17th STOC*, 1985, pp. 291–304.

[GS] Goldwasser, S., and M. Sipser, Private Coins vs. Public Coins in Interactive Proof Systems, *Proc. 18th STOC*, 1986, pp. 59–68.

[H] Hastad, J., Psuedo-random Generators Under Uniform Assumptions, *Proc. 22nd STOC*, 1990, pp. 395–404.

[ILL] Impagliazo, R., L. A. Levin, and M. Luby, Pseudorandom Generation from One-Way Functions, *Proc. 21st STOC*, 1989, pp. 12–24.

[IY] Impagliazo, R., and M. Yung, Direct Minimum-Knowledge Computations, *Advances in Cryptology—Crypto 87 (Proceedings)*, C. Pomerance (ed.), pp. 40–51, Lecture Notes in Computer Science, Vol. 293, Springer-Verlag, Berlin, 1987.

[Ka] Kaliski, B. S., Elliptic Curves and Cryptography: A Pseudorandom Bit Generator and Other Tools. Ph.D. Thesis, MIT/LCS/TR-411, Massachusetts Institute of Technology, 1988.

[Kuc] Kucera, L., Canonical Labeling of Regular Graphs in Linear Average Time, *Proc. 28th FOCS*, 1987, pp. 271–279.

[Kus] Kushilevitz, E., Perfect Zero-Knowledge Proofs, Master Thesis, Technion, 1989 (in Hebrew). A translation in English of the subsection concerning the parallel execution of the basic protocol is available from the author.

[N] Naor, M., Bit Commitment Using Pseudorandomness, *Advances in Cryptology—Crypto 89 (Proceedings)*, G. Brassard, (ed.), pp. 128–136, Lecture Notes in Computer Science, Vol. 435, Springer-Verlag, Berlin, 1990.

[Od] Odlyzko, A., Discrete Logarithm in Finite Fields and Their Cryptographic Significance, *Proc. Eurocrypt 84*, pp. 224–314, Lecture Notes in Computer Science, Vol. 209, Springer-Verlag, Berlin, 1985.

[Or] Oren, Y., On the Cunning Power of Cheating Verifiers: Some Observations about Zero-Knowledge Proofs, *Proc. 28th FOCS*, 1987, pp. 462–471.

[RS] Rosser, J., and L. Schoenfield, Approximate Formulas for Some Functions of Prime Numbers, *Illinois J. Math.*, Vol. 6, 1961, pp. 64–94.

[S] Shamir A., IP = PSPACE, *Proc. 31st FOCS*, 1990, pp. 11–15.

[TW] Tompa, M., and H. Woll, Random Self-Reducibility and Zero-Knowledge Interactive Proofs of Possession of Information, *Proc. 28th FOCS*, 1987, pp. 472–482.

[Y] Yao, A. C., Theory and Applications of Trapdoor Functions, *Proc. 23rd FOCS*, 1982, pp. 80–91.