AN IMPROVED PROTOCOL FOR DEMONSTRATING POSSESSION OF DISCRETE LOGARITHMS AND SOME GENERALIZATIONS

David Chaum
Jan-Hendrik Evertse
Jeroen van de Graaf

Centre for Mathematics and Computer Science
Kruislaan 413 1098 SJ Amsterdam The Netherlands

Abstract:
A new protocol is presented that allows A to convince B that she knows a solution to the Discrete Log Problem—i.e. that she knows an x such that $\alpha^x \equiv \beta \pmod{N}$ holds—without revealing anything about x to B. Protocols are given both for N prime and for N composite.

We also give protocols for extensions of the Discrete Log problem allowing A to show possession of:

— multiple discrete logarithms to the same base at the same time, i.e. knowing $x_1, \ldots, x_K$ such that $\alpha^{x_1} \equiv \beta_1, \ldots, \alpha^{x_K} \equiv \beta_K$;

— several discrete logarithms to different bases at the same time, i.e. knowing $x_1, \ldots, x_K$ such that the product $\alpha_1^{x_1} \alpha_2^{x_2} \cdots \alpha_K^{x_K} \equiv \beta$;

— a discrete logarithm that is the simultaneous solution of several different instances, i.e. knowing x such that $\alpha_1^x \equiv \beta_1, \ldots, \alpha_K^x \equiv \beta_K$.

We can prove that the sequential versions of these protocols do not reveal any "knowledge" about the discrete logarithm(s) in a well-defined sense, provided that A knows (a multiple of) the order of α.

1. Introduction

Consider the following problem:

- Alice (party A) knows a solution to the **Discrete Log (DL)** problem: for particular α, β and N, she knows the exponent x such that $\alpha^x \equiv \beta \pmod{N}$ holds.
- Alice wants to convince Bob (party B) that she knows x.

D. Chaum and W.L. Price (Eds.): Advances in Cryptology - EUROCRYPT '87, LNCS 304, pp. 127-141, 1988.

- Alice is not willing to reveal the value of x.
- Bob accepts an exponentially small chance that Alice is cheating, i.e. that she pretends to know an x but in fact does not. More precisely, the probability that Alice succeeds in cheating without being detected by Bob, is 2^{-T}, where T is proportional to the time and space required.

This paper presents a protocol which solves this problem, both for the cases that N is a prime and that $N = P_1 P_2$, where P_1 and P_2 are prime numbers of roughly the same size. In the second case, it is assumed that A knows the factorization of N. When A does not know this factorization, however, our protocol is still of interest, since given α and N she can choose $x \in \{1, \ldots, N-1\}$ at random and then compute β simply by exponentiation (or a third party could supply A with x and β). It is assumed that B has only polynomial (in log N) computational power, whereas no restrictions are imposed on A's computational resources. No probabilistic polynomial time algorithm is known for finding x given α, β and N, if N is a prime or a composite that is difficult to factor.

In [CEGP86] protocols were presented that solve the same problem. Compared to those protocols, the basic protocol presented here is perhaps easier to understand, to use, and to generalize. The existence of a protocol with the same functionality is implied by general results of [BrCr86], [Ch86] and [GMW86]. However, these protocols are not very useful in practice. In [Ch87] efficient protocols that solve this problem are needed; this was the major motivation for our research.

We also present protocols for proving possession of a solution to some generalizations of the Discrete Log problem:

(1) **Multiple Discrete Log (MDL):**
A shows to B that, given α and $\beta_1, \ldots, \beta_K$, she knows $x_1, \ldots, x_K$ such that $\alpha^{x_1} \equiv \beta_1, \ldots, \alpha^{x_K} \equiv \beta_K$. This protocol is more efficient than applying the basic DL-protocol for the pairs $(x_1, \beta_1), \ldots, (x_K, \beta_K)$ while it gives B the same probability of catching a cheating A. When a third party creates the x_i's at random and supplies A with the x_i's and β_i's, this protocol also offers the possibility to use DL as the basis for an authentication scheme in a way similar to Fiat & Shamir [FiSh86], whose scheme is based on the difficulty of factoring.

(2) **Relaxed Discrete Log (RDL):**
A shows to B that, given $\alpha_1, \ldots, \alpha_K$ and β, she knows $x_1, \ldots, x_K$ such that $\alpha_1^{x_1} \alpha_2^{x_2} \cdots \alpha_K^{x_K} \equiv \beta$.

(3) **Simultaneous Discrete Log (SDL):**
A shows to B that, given $\alpha_1, \ldots, \alpha_K$ and $\beta_1, \ldots, \beta_K$, she knows x such that $\alpha_i^x \equiv \beta_i$ for $i = 1, \ldots, K$.

The Discrete Log problem is stated above in Z_N^* (the multiplicative group of residue classes modulo N of integers coprime with N) with N prime or composite. However, the Discrete Log problem can be stated in any finite group: let G be a finite group, $<\alpha>$ the subgroup generated by $\alpha \in G$, and $\beta \in <\alpha>$; then find x such that $\alpha^x = \beta$. The protocols presented in this paper are feasible in any group G in which both A and B can apply the group operation in an efficient way, e.g. in time polynomial in the logarithm of the order of G. (For the RDL-protocol we also have to assume that G is commutative). The properties of the DL-protocol over Z_N^* which are proved in this paper (namely that it allows A to convince B with high probability that she knows the discrete logarithm of β with respect to α without revealing any knowledge about that discrete logarithm) remain true for the DL-protocol over any group G, such that A knows (a positive multiple of) the order of α in G and B knows a "good" approximation of (a positive multiple of) the order of α, i.e. if m is some multiple of the order of α then B knows an integer m' such that $|m - m'| \leqslant m^c$, where c is some number with $0 \leqslant c < 1$. For instance, if $G = Z_N^*$, then B knows the exact order of G if N is a prime, while if $N = P_1 P_2$ with P_1 and P_2 primes of order $O(N^{1/2})$, then G has order $\phi(N) = (P_1 - 1)(P_2 - 1)$, B knows N and $|N - \phi(N)| = O(N^{1/2})$. The DL-protocol can be used also if B does not know a good approximation for (a multiple of) the order of α; however, B may be able to obtain such an approximation by examining the messages which he receives from α while participating in the protocol. Further, with a slight modification, the DL-protocol is still feasible if A does not know a multiple of the order of α in G, but then the protocol leaks information about x.

Of course, these protocols are of interest only if no efficient algorithm for computing the Discrete Log in G exists. Apart from the case $G = Z_N^*$, with N prime or composite, we can take the K-fold direct product of Z_N^*, giving rise to the Simultaneous Discrete Log protocol, or the set of points of an elliptic curve over $GF(P)$ for some prime P, imposed with the usual group structure. It was argued in [Mi85] that discrete logarithms in the group of points of an elliptic curve over $GF(P)$ might be even harder to compute than "ordinary" discrete logarithms.

For describing the protocols, we use the same protocol notation throughout the paper. The meaning of this notation is straightforward; only the next few things might need explanation:

— T is the **security parameter**, agreed upon before the protocol starts. Increasing T reduces A's chance of successfully cheating exponentially, but increases the amount of communication and computation only linearly.

— In the zeroth step of the protocol, A and B agree on α, β and N.

— If not indicated otherwise, the expressions appearing in the protocol have to be reduced modulo N.

— By $a :\equiv$ *expression* (mod M) we mean that the expression at the right-hand side must be computed and reduced modulo M and that the resulting value is assigned to a; if $M = N$ we omit the suffix "(mod N)".

— $e \in_R S$ indicates that an element e is chosen at random from the set S, i.e. all elements of S have an equal probability of being chosen and that the choice is independent of all previous events.

- In some steps of the protocol a party checks if a particular equality holds; this is denoted as: check $a \stackrel{?}{=} b$. If the check fails, cheating is detected and the protocol halts.
- Expressions shown on the left or right are known to the corresponding party only, and are secret from the other party.
- A party cannot learn anything about the computations that are done by the other party, except from the messages which (s)he received from that party.

2. The basic protocol: Discrete Log

Instance: $N,\ \alpha \in Z_N^*,\ \beta \in <\alpha>$
Solution: x such that $\alpha^x \equiv \beta \pmod N$

In order for the protocol to make sense, one has to assume that there are no efficient (polynomial in log N time) algorithms to compute discrete logarithms modulo N for N prime or composite. It is generally believed, that for large primes N satisfying certain weak restrictions, it is infeasible to compute discrete logarithms in Z_N^*. In this paper we assume that computing discrete logarithms is also hard when N is a product of two primes that is difficult to factor. Our motivation behind this assumption is that any fast method to compute for each pair $\alpha \in Z_N^*$ and $\beta \in <\alpha>$ an integer x with $\alpha^x \equiv \beta \pmod N$, enables one to efficiently find the factorization of N with high probability. Indeed, choose γ at random from Z_N^* and pick a "probable prime" p between N and $2N$. Compute $\alpha :\equiv \gamma^{2p}$, $\beta :\equiv \gamma^2$. Then with high probability, p is a prime number coprime with $\phi(N)$, whence $\beta \in <\alpha>$. Suppose that the discrete log algorithm computes an x with $\beta \equiv \alpha^x$. Then $\gamma^{2(px-1)} \equiv 1$, hence γ^{px-1} is a square root of 1. With 50% chance, this square root is not equal to 1 or -1 and yields the factorization of N. It is in fact possible to prove the following stronger (and from a cryptographic point of view more convincing) statement. Let N be a given product of two large primes and suppose that there is a random polynomial time algorithm (i.e. an algorithm whose running time is polynomial in the length of the input and which can do unbiased coinflips) with the following property: when the algorithm is given the pair α, β as input, where α is uniformly distributed on Z_N^* and β is uniformly distributed on $<\alpha>$, then the probability that that algorithm outputs an integer x with $\alpha^x \equiv \beta$ is at least $1/Q(\log N)$ for some polynomial Q. Then there is a random polynomial time algorithm that outputs the factorization of N with probability at least ½. We do not work this out here.

We develop the protocols simultaneously for both the cases N prime and N composite, and point out the differences. If N is composite, we assume that A knows its factorization.

Protocol 1: Discrete Log: $\alpha^x \equiv \beta \pmod{N}$

	A		*B*
Step 0:		α, β, N $\longleftrightarrow$	
Repeat T times:			
	$r \in_R \{1, \ldots, \phi(N)\}$ $\gamma :\equiv \alpha^r$		
Step 1:		γ $\longrightarrow$	
			$b \in_R \{0, 1\}$
Step 2:		b $\longleftarrow$	
	$y :\equiv r + bx \pmod{\phi(N)}$		
Step 3:		y $\longrightarrow$	
			check $\alpha^y \stackrel{?}{\equiv} \gamma\beta^b$

2.1. Remarks about the underlying mathematical model

Our purpose is to prove that Protocol 1 (and the other protocols that will be described in this paper) have the following properties:

— *correctness:* even a cheating party A is unable (or with a very small chance able) to send messages to B satisfying all of B's checks;

— *security:* B cannot obtain any useful "knowledge" about the discrete logarithm from the protocol other than from the initializing information α, β and N, even if he cheats.

In this subsection we explain more precisely what is meant by "cheating" and what it means that no knowledge is revealed. In the remainder of this paper we assume that the computational power of B is polynomially bounded in $\log N$. For the computational power of A, we do not make any assumption since it does not matter in our arguments whether A's computational power is polynomially bounded or not.

We say that A cheats if she constructs her messages by means of some probabilistic algorithm, in another way than that described in the protocol. For instance, if A does not know the discrete logarithm, then she could try to construct her messages in such a way that they still satisfy B's checks. B cheats if he generates his bits in step 2 using a random polynomial time algorithm that does not choose them at random.

In several papers, e.g. [GMR85], [BKP85], [GMW86], and [CEGP86], it was argued that the security of a protocol can be proved by showing the existence of a random polynomial time "simulator" that simulates the interaction between A and B using as input only what B knows at the beginning of the protocol. For convenience of the reader, we explain below the notion of such a simulator, and why its existence suffices.

Informally speaking, we would like to prove that in whatever way B tries to cheat, the data he obtains during his participation in the protocol do not help him find a solution to any equation (*) $f(\alpha,\beta,N,z)=0$ in the unknown z. Before the protocol starts, B gets α, β and N. In step 1, B gets $\gamma \in Z_N^*$ from A. In step 2, B generates a bit b. If B cheats, then he generates b in another way than just choosing it at random; he might use all messages that he computed or received before (in the first round of the protocol these are only N, α, β, and γ). During the execution of the algorithm that produces b, B might obtain intermediate results, some of which he would like to store for later purposes; let $\mathbf{b}$ comprise the intermediate results stored by B. Finally, in step 3, B receives an integer y from A such that $\alpha^y \equiv \gamma\beta^b$. Thus B gets a tuple $(\gamma,\mathbf{b},b,y)$. After steps 1, 2, and 3 have been executed T times, B has obtained a tuple $W_B = (\gamma_1,\mathbf{b}_1,b_1,y_1, \ldots, \gamma_T,\mathbf{b}_T,b_T,y_T)$ containing all data obtained by B during his participation in the protocol. Note that W_B is stochastic, and that its probability distribution depends on the initializing information $I_A=(\alpha,\beta,N,x)$.

Suppose that B has a probabilistic algorithm M_f that computes a solution to equation (*) with some positive probability. Further, suppose that there is a "simulator" S, with small (polynomial in $\log N$) running time, which produces a tuple W_B^* with about the same probability distribution as W_B, on input $I_A^*=(\alpha,\beta,N)$. This simulator may depend on B's way of cheating. Let M_f^* be the algorithm that first computes W_B^* in the same way as S, on input I_A^*, and then computes a solution to (*) by applying M_f to I_A^* and W_B^*. M_f^* outputs a solution to (*) with about the same probability as M_f (since W_B and W_B^* have about the same probability distribution) and M_f^* has about the same running time as M_f. This shows that the protocol does not reveal any useful knowledge to B: algorithm M_f when input the data gathered by B during the performance of the protocol does not output a solution to (*) faster or with higher probability than algorithm M_f^* when input the initialization data I_A^* only. Hence in order for the protocol to be secure, it suffices that there is a simulator with small running time for each way of cheating by B.

It is possible to give the notion of a simulator, informally described above, a formal meaning similar to [GMR85], [BKP85] or [CEGP86]. We assume that the reader is familiar with the formal definition of a protocol and with the underlying computational model, as described in [BKP85]. We use a slightly different model that is briefly described below.

We consider cryptographic protocols with two parties, a "prover" A and a "verifier" B. Both A and B use probabilistic Turing machines T_A and T_B, respectively, with a work tape, a random tape and a "mailbox". The machines use the same alphabet Σ. Each machine can read only from its own work tape, random tape, and mailbox, but it can write on its own work tape as well as on the other machine's mailbox. Each step executed by a machine is determined by the machine's state and the contents of its three tapes, and does not depend on the other machine's state. Whenever a machine has to send a message to the other machine, it copies that message from its own work tape to the other machine's mailbox; then the other machine may copy this message from its mailbox to its work tape. For convenience we assume that the machines do not run simultaneously. Thus after a machine has written a complete message string on the other machine's mailbox, it stops and is reactivated again only after it has received a message from the other machine.

Before the protocol starts, both machines are in a fixed initialization state, and the work tapes of these machines are filled with certain initialization data I_A^*. Further, T_A's work tape contains the secret x. Put $I_A = (I_A^*, x)$; then I_A is a string of length l, say, over Σ. Further, in the beginning both random tapes are filled with an infinite number of symbols, each uniformly chosen from Σ. At the end of the protocol, both machines are supposed to be in an end state. We suppose that the number of steps performed by T_B between the initialization state and the end state is bounded above by a polynomial in l; for our purposes it does not matter whether or not the number of steps executed by T_A between the initialization state and the end state is polynomially bounded in l.

Denote by W_B the contents of T_B's work tape in the end state. W_B contains all data stored by T_B while the protocol was running; these data might contain the messages sent and received by T_B and some final or intermediate results of T_B's computations. Because of the use of random tapes, W_B is a stochastic variable whose probability distribution depends on I_A. We assume that for each I_A, W_B assumes its values in some enumerable set Ω; let $\mathbf{P}_{I_A}$ denote the probability distribution of W_B on Ω. An A-simulator, based on machine T_B, is defined as a probabilistic Turing machine which produces a tuple W_B^* with almost the same probability distribution as W_B (but depending only on I_A^*); more precisely, if $\mathbf{P}_{I_A^*}$ denotes the probability distribution of W_B^* then for each I_A with sufficiently large length l we have

$$\sum_{\omega \in \Omega} \left| \mathbf{P}_{I_A}(W_B = \omega) - \mathbf{P}_{I_A^*}(W_B^* = \omega) \right| \leqslant C^{-l},$$

where C is some absolute constant with $C > 1$.

2.2. Correctness and security of Protocol 1

In this subsection we prove that Protocol 1 is correct and secure. In the theorem below we assume that T is polynomially bounded. (By "polynomial" we always mean polynomial in $\log N$.)

Theorem 1 .
(a) If B does not cheat, and if A does not know the discrete logarithm x, then any cheating by A in Protocol 1 is detected by B with probability $\geqslant 1-2^{-T}$.
(b) If A does not cheat, then for any random polynomial time machine used by B in Protocol 1, there exists a polynomial time A-simulator.

Proof:
(a) Correctness: If A does not know x, then each time that step 3 is executed, she is unable to send the proper answer to B in at least one of the cases $b = 0$ or $b = 1$. Hence, in each round of the protocol, she will be caught with probability at least ½. Thus B will detect that A does not know x with probability at least $1 - 2^{-T}$.
(b) Security (sketch): Let T_B be the random polynomial time machine used by B. Suppose for the moment that the number of rounds T is equal to 1. We have $I_A = (\alpha,\beta,N,x)$, $I_A^* = (\alpha,\beta,N)$ and $W_B = (\gamma,\mathbf{b},b,y)$ where: γ is the message received by B in step 1; b is the bit computed by T_B in step 2, using γ; $\mathbf{b}$ comprises the intermediate steps in the computation of b stored by T_B; and y is the integer received by B in step 3, satisfying $\alpha^y \equiv \gamma\beta^b \pmod N$. Then the polynomial time A-simulator is described as follows (all expressions have to be reduced modulo N):

Repeat at most $L := \log N / \log 2$ times:
(1) choose c at random from $\{0, 1\}$
(2) choose y at random from $\{0, \ldots, N-2\}$
(3) compute $\gamma :\equiv \alpha^y \beta^{-c}$
(4) compute $b \in \{0, 1\}$ using T_B; let $\mathbf{b}$ comprise the saved intermediate results
(5) if $b = c$ then output $W_B^* = (\gamma,\mathbf{b},b,y)$
until $b = c$
If $b \neq c$ in all L executions of steps (1)-(5), then output W_B^* = "badluck"

Note that this simulator has polynomial running time.

Suppose first that N is a prime number and consider one execution of steps (1)-(5) described above. In this execution, γ is uniformly distributed over $<\alpha>$, and γ and c are mutually independent. Further, in the computation of b, only γ is used, hence b is also independent of c. Therefore, $b = c$ with probability ½. This implies that the probability that $b = c$ in at least one of L executions of steps (1)-(5) is at least $1 - N^{-1}$. Note that $\alpha^y \equiv \gamma\beta^b$. Let Ω be the set of values which can be assumed by W_B^*, including the message "badluck". It is easy to verify that for each $\omega \in \Omega$ with $\omega \neq$ "badluck" we have

$$\mathbf{P}_{I_A^*}(W_B^* = \omega \mid W_B^* \neq \text{badluck}) = \mathbf{P}_{I_A}(W_B = \omega).$$

Together with the fact that $\mathbf{P}_{I_A^*}(W_B^* = \text{badluck}) \leqslant N^{-1}$ this shows that

$$S := \sum_{\omega \in \Omega} \left| \mathbf{P}_{I_A}(W_B = \omega) - \mathbf{P}_{I_A^*}(W_B^* = \omega) \right| \leqslant 2N^{-1}.$$

Since the length of I_A is proportional to $\log N$, this implies part (b) of Theorem 1 if N is a prime.

Now suppose that $N = P_1 P_2$ where P_1 and P_2 are primes of order $N^{1/2}$. Then $N-1$ is not a multiple of the order of α in Z_N^*, hence the number γ computed in step 3 is not uniformly distributed over $<\alpha>$. However, all arguments given above remain valid if we consider conditional probabilities given that $0 \leqslant y \leqslant \phi(N) - 1$. Using that $\mathbf{P}_{I_A^*}(\phi(N) \leqslant y \leqslant N-2) = O(N^{-1/2})$, it follows that S is bounded above by $O(N^{-1/2})$.

If $T > 1$, the simulator described above has to be repeated T times. This increases the running time by a factor T, and S by a factor $\leqslant T$. But since T is bounded above by a polynomial in $\log N$, this completes the proof of Theorem 1. □

3. Generalization 1: Multiple Discrete Log

Instance: N, $\alpha \in Z_N^*$, $\beta_1, \ldots, \beta_K \in <\alpha>$

Solution: $x_1, \ldots, x_K$ such that $\alpha^{x_1} \equiv \beta_1 \pmod N, \ldots, \alpha^{x_K} \equiv \beta_K \pmod N$

Protocol 2: Multiple Discrete Log: $\alpha^{x_1} \equiv \beta_1 \pmod N, \ldots, \alpha^{x_K} \equiv \beta_K \pmod N$

	A		***B***
Step 0:		$\xleftrightarrow{\alpha, \beta_1, \ldots, \beta_K, N}$	
Repeat T times:			
	$r \in_R \{1, \ldots, \phi(N)\}$ $\gamma :\equiv \alpha^r$		
Step 1:		$\xrightarrow{\gamma}$	
			$b_i \in_R \{0, 1\}$,
Step 2:		$\xleftarrow{b_1, \ldots, b_K}$	
	$y :\equiv r + b_1 x_1 + \cdots + b_K x_K \pmod{\phi(N)}$		
Step 3:		$\xrightarrow{y}$	
			check $\alpha^y \stackrel{?}{\equiv} \gamma \beta_1^{b_1} \cdots \beta_K^{b_K}$

We assume that T and 2^K are bounded above by some polynomial in $\log N$.

Theorem 2 .
(a) If B does not cheat, and if A does not know at least one of the discrete logarithms $x_1, \ldots, x_K$, then any cheating by A in Protocol 2 is detected by B with probability $\geqslant 1-2^{-T}$.
(b) If A does not cheat, then for any random polynomial time machine used by B in Protocol 2, there exists a polynomial time A-simulator.

Proof:
(a) Correctness: Consider one round of the protocol, consisting of steps 1, 2, and 3. By assumption, A does not know the discrete logarithm of at least one β_i (with respect to α). Hence for whatever γ she computes in step 1, she is not able to compute the discrete logarithm of $\gamma \beta_1^{b_1} \cdots \beta_K^{b_K}$ for at least one vector $(b_1, \ldots, b_K) \in \{0, 1\}^K$. Together with the lemma below this implies that, in each round, A is caught cheating with probability at least ½. Hence her cheating is detected by B with probability at least $1-2^{-T}$.

Lemma: *Suppose that A does not know the discrete logarithm of $\gamma(\vec{b}) :\equiv \gamma \beta_1^{b_1} \cdots \beta_K^{b_K} \pmod N$ for at least one vector $\vec{b} = (b_1, \ldots, b_K) \in \{0, 1\}^K$. Then she does not know the discrete logarithm of $\gamma(\vec{b})$ for at least half the vectors $\vec{b} \in \{0, 1\}^K$.*

Proof: We proceed by induction on K. For $K = 1$ the lemma is trivial. Suppose now that the lemma is true for $K = L - 1$, where $L \geqslant 2$ (induction hypothesis). We shall prove the lemma for $K = L$. We distinguish three cases. In what follows, $\vec{b}$ always denotes a vector $(b_1, \ldots, b_L) \in \{0, 1\}^L$, and $\gamma(\vec{b})$ has the same meaning as above with L replacing K.

In the first case, A knows the discrete logarithms of all the products $\gamma(\vec{b})$ with $b_L = 0$. Thus, she cannot know the discrete logarithm of β_L. Hence she cannot form the discrete logarithm of any product $\gamma(\vec{b})$ with $b_L = 1$.

In the second case, A knows the discrete logarithm of each product $\gamma(\vec{b})$ with $b_L = 1$. Then, by the same argument as in case 1, it follows that A cannot form the discrete logarithm of any product $\gamma(\vec{b})$ with $b_L = 0$.

In the last case, A does not know the discrete logarithm of at least one of the products $\gamma(\vec{b})$ with $b_L = 0$ and also not the discrete logarithm of at least one of the products $\gamma(\vec{b})$ with $b_L = 1$. Then by the induction hypothesis, she does not know the discrete logarithm of at least half the products $\gamma(\vec{b})$ with $b_L = 0$ and also, by the induction hypothesis with $\gamma \beta_L$ instead of γ, she does not know the discrete logarithm of at least half the products $\gamma(\vec{b})$ with $b_L = 1$.

We conclude that in each of the three cases A cannot know the discrete logarithm of at least half the products $\gamma(\vec{b})$. This completes the induction step. □

(b) Security. The proof is essentially the same as that of Theorem 1, part (b). We only describe the A-simulator. B uses machine T_B.

For $i = 1$ to T:
  repeat at most $L' := \log N / \log(1 - 2^{-K})$ times:
    choose $\vec{c}_i = (c_{1i}, \ldots, c_{Ki})$ at random from $\{0, 1\}^K$
    choose y_i at random from $\{0, \ldots, N-2\}$
    compute $\gamma_i :\equiv \alpha^{y_i} \beta_1^{-b_{1i}} \cdots \beta_K^{-b_{Ki}}$
    compute $\vec{b}_i \in \{0, 1\}^K$ with T_B; let $\mathbf{b}_i$ comprise the intermediate results of T_B's computations
    if $\vec{b} = \vec{c}$ then output $(\gamma_i, \mathbf{b}_i, \vec{b}_i, y_i)$
  until $\vec{b} = \vec{c}$
  if $\vec{b}_i \neq \vec{c}_i$ in all L' iterations, then output "badluck"
If not at least once "badluck" then output $W_B^* = (\gamma_1, \mathbf{b}_1, \vec{b}_1, y_1, \ldots, \gamma_K, \mathbf{b}_K, \vec{b}_K, y_K)$

Note that the running time of this simulator is proportional to T and 2^K, but by assumption these numbers are bounded above by some polynomial in $\log N$. □

Remark 1. It is possible to use Protocol 2 as an interactive "identification scheme," a concept introduced by Fiat and Shamir [FiSh86]. Suppose that not A, but some mutually trusted "center" generates the x_i's at random, supplies these to A (but to nobody else) and stores the corresponding β_i's in some public directory. Then A can identify herself to B by showing that she knows the discrete logarithms of the β_i's without revealing any knowledge about their values, using Protocol 2. Thus, the data obtained from his interaction with A will not enable B to identify himself to a third party as A. The Fiat-Shamir scheme uses a public composite number, whose factorization is known only to the center. In that scheme, the β_i's for a user A are squares modulo that composite, constructed by the center, and A has to convince B that she possesses square roots of these β_i's. Contrary to our scheme used with a prime modulus, in the Fiat-Shamir scheme the center must keep some trapdoor information secret (namely the factorization of the modulus). On the other hand, Fiat and Shamir argued that there scheme allows the center to form the β_i's of some user A by applying some public function to A's name and address or the like. Thus, any verifier B can compute the β_i's by himself and they do not have to be stored in a public file. The function that is used to construct the β_i's should be such that only the center, knowing the factorization of the modulus, is able to compute a square root of some output of the function. However, it is currently not known how to prove that any such public function prevents people from constructing names for which they can find corresponding square roots themselves. The scheme of Fiat and Shamir is more efficient than ours, because it requires only squaring whereas our scheme requires exponentiations of $\log N$-bit numbers.

Remark 2. If we assume that not 2^K but K is bounded above by a polynomial in $\log N$, then the running time of the simulator described above is not polynomial any more since it is proportional to 2^K. It seems impossible to construct a simulator whose running time depends only polynomially on K for *each* machine used by B, since B might generate its bits by some one-way function. However, there does exist a simulator (described below) for the machine that chooses the bits to be sent from B to A uniformly from $\{0, 1\}$. In order to prevent B from choosing the bits to be sent to A not uniformly, one could modify the protocol so that the bits are chosen not by B alone, but by A and B together, using a coin flipping protocol like that in [Bl82]. The protocol thus modified is called "verifier-passive" (cf. [CEGP86]) because B can do nothing but checking that A sends the correct answers. The simulator is described below:

For $i=1$ to T:
  choose $\vec{b_i} = (b_{1i}, \ldots, b_{Ki})$ at random from $\{0, 1\}^K$
  choose y_i at random from $\{0, \ldots, N-2\}$
  compute $\gamma_i :\equiv \alpha^{y_i} \beta_1^{b_{1i}} \cdots \beta_K^{b_{Ki}}$
Output $W_B^* = (\gamma_1, \vec{b}_1, y_1, \ldots, \gamma_T, \vec{b}_T, y_T)$

4. Generalization 2: Relaxed Discrete Log

Instance: $N,\ \alpha_1, \ldots, \alpha_K \in Z_N^*,\ \beta \in Z_N^*$
Solution: $x_1, \ldots, x_K$ such that $\alpha_1^{x_1} \cdots \alpha_K^{x_K} \equiv \beta \pmod N$

It is easy to see that if there exists an efficient algorithm which computes a solution to the Relaxed Discrete Log problem for each instance, then there is also a fast way to compute discrete logarithms for each possible instance: in order to find the discrete logarithm of β with respect to α one has merely to solve the Relaxed Discrete Log problem for the instance $N, \alpha, 1, \ldots, 1, \beta$. It is possible to prove the following stronger result. Let N, K be given integers such that N is either a prime or the product of two primes and that K is bounded above by a polynomial in $\log N$ and suppose that there exists a random polynomial (in $\log N$) time algorithm with the following property: if $\alpha_1, \ldots, \alpha_K$ and β are given as input to the algorithm, where $\alpha_1, \ldots, \alpha_K$ are uniformly distributed over Z_N^* and β is uniformly distributed over $<\alpha_1, \ldots, \alpha_K>$, then that algorithm outputs integers $x_1, \ldots, x_K$ such that $\alpha_1^{x_1} \cdots \alpha_K^{x_K} \equiv \beta \pmod N$ with probability at least $1/Q(\log N)$ for some polynomial Q. Then there is a random polynomial time algorithm that computes for each pair $\alpha \in Z_N^*$ and $\beta \in <\alpha>$ with probability $\geqslant ½$ an integer x such that $\alpha^x \equiv \beta \pmod N$. This statement is not proved here.

Protocol 3: Relaxed Discrete Log: $\alpha_1^{x_1} \cdots \alpha_K^{x_K} \equiv \beta \pmod N$

	A		***B***
Step 0:		$\alpha_1, \alpha_2, \ldots, \alpha_K, \beta, N$ $<\!\!-\!\!-\!\!-\!\!-\!\!-\!\!-\!\!-\!\!-\!\!>$	
Repeat T times:			
	$r_1, \ldots, r_K \in_R \{1, \ldots, \phi(N)\}$ $\gamma_1 :\equiv \alpha_1^{r_1}, \ldots, \gamma_K :\equiv \alpha_K^{r_K}$		
Step 1:		$\gamma_1, \ldots, \gamma_K$ $-\!\!-\!\!-\!\!-\!\!-\!\!-\!\!-\!\!-\!\!>$	
			$b \in_R \{0, 1\}$
Step 2:		b $<\!\!-\!\!-\!\!-\!\!-\!\!-\!\!-\!\!-\!\!-$	
	$y_i :\equiv r_i + bx_i \pmod{\phi(N)}$ for $1 \leqslant i \leqslant K$		
Step 3:		$y_1, \ldots, y_K$ $-\!\!-\!\!-\!\!-\!\!-\!\!-\!\!-\!\!-\!\!>$	
			check $\alpha_1^{y_1} \cdots \alpha_K^{y_K} \equiv$ $\equiv \gamma_1 \cdots \gamma_K \beta^b$

If K and T are bounded above by a polynomial in $\log N$ then we have:

Theorem 3 .
(a) If B does not cheat and if A does not know at least one of $x_1, \ldots, x_K$, *then any cheating by A in Protocol 3 is detected by B with probability* $\geqslant 1-2^{-T}$.
(b) If A does not cheat, then for any random polynomial time machine used by B in Protocol 3 there exists a polynomial time A-simulator.

The proof of this result is essentially the same as that of Theorem 1 and we do not give it here.

5. Generalization 3: Simultaneous Discrete Log

Instance: $N, \alpha_1, \ldots, \alpha_K, \beta_1, \ldots, \beta_K \in Z_N^*$
Solution: x such that $\alpha_1^x \equiv \beta_1 \pmod N), \ldots, \alpha_K^x \equiv \beta_K \pmod N$

Protocol 4: Simultaneous Discrete Log: $\alpha_1^x \equiv \beta_1 \pmod N, \ldots, \alpha_K^x \equiv \beta_K \pmod N$

	A		***B***
Step 0:		$\alpha_1, \ldots, \alpha_K, \beta_1, \ldots, \beta_K, N$ $<\!\!\longrightarrow$	
Repeat T times:			
	$r \in_R \{1, \ldots, \phi(N)\}$ $\gamma_1 :\equiv \alpha_1^r, \ldots, \gamma_K :\equiv \alpha_K^r$		
Step 1:		$\gamma_1, \ldots, \gamma_K$ $\longrightarrow$	
			$b \in_R \{0, 1\}$
Step 2:		b $\longleftarrow$	
	$y :\equiv r + bx \pmod{\phi(N)}$		
Step 3:		y $\longrightarrow$	
			check $\alpha_i^y \stackrel{?}{=} \gamma_i \beta_i^b$ for $1 \leqslant i \leqslant K$

We assume again that both K and T are bounded above by a polynomial in $\log N$.

Theorem 4 .
(a) If B does not cheat and if A does not know x, then any cheating by A in Protocol 4 is detected by B with probability $\geqslant 1-2^{-T}$.
(b) If A does not cheat then for any random polynomial time machine used by B in Protocol 4, there exists a polynomial time A-simulator.

The proof of this theorem is essentially that of Theorem 1.

Remark. From an algebraic point of view, this protocol is just Protocol 1 applied to the K-fold direct product $(Z_N^*)^K$, with $\alpha = (\alpha_1, \ldots, \alpha_K)$ and $\beta = (\beta_1, \ldots, \beta_K)$.

Acknowledgements

We would like to thank René Peralta and Bert den Boer for their valuable suggestions.

References

[BKP85] R. Berger, S.Kannan, and R. Peralta, "A Framework for the Study of Cryptographic Protocols," *Proc. CRYPTO 85*, pp. 87-103 H.C. Williams, ed., Lecture Notes in Computer Science 218, Springer Verlag, Berlin etc., (1986).

[Bl82] M. Blum, "Coin Flipping by Telephone," *Proc. IEEE COMPCON*, pp. 133-137, (1982).

[BrCr86] G. Brassard, and C. Crépeau, "Zero-Knowledge Simulation of Boolean Circuits," *Proc. CRYPTO 86*, pp. 223-233, A.M. Odlyzko, ed., Lecture Notes in Computer Science 263, Springer Verlag, Berlin etc., (1987).

[Ch86] D. Chaum, "Demonstrating that a Public Predicate can be Satisfied Without Revealing Any Information About How," *Proc. CRYPTO 86*, pp. 195-199.

[Ch87] D. Chaum, "Blinding for unanticipated signatures," *To appear in proc. EUROCRYPT 87.*

[CEGP86] D. Chaum, J.-H. Evertse, J. van de Graaf, and R. Peralta, "Demonstrating possession of a discrete logarithm without revealing it," *Proc. CRYPTO 86*, pp. 200-212.

[FiSh86] A. Fiat, and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," *Proc. CRYPTO 86*, pp. 186-194.

[GMR85] S. Goldwasser, S.Micali, and C. Rackoff, "The Knowledge Complexity of Interactive Proof Systems," *Proc. 17th Annual ACM Symp. on Theory of Computing*, pp. 291-304, (1985).

[GMW86] O. Goldreich, S. Micali, and A. Wigderson, "How to Prove all NP-statements in Zero-Knowledge, and a Methodology of Cryptographic Protocol Design," *Proc. CRYPTO 86*, pp. 171-185.

[Mi85] V. Miller, "Elliptic curves and cryptography," *Proc. CRYPTO 85*, pp. 417-428.