DEMONSTRATING POSSESSION OF A DISCRETE LOGARITHM WITHOUT REVEALING IT

David Chaum
Jan-Hendrik Evertse[*]
Jeroen van de Graaf
René Peralta[**]

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

Abstract: Techniques are presented that allow 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 prove these protocols secure under a formal model which is of interest in its own right. We also show how A can convince B that two elements α and β generate the same subgroup in Z_N^*, without revealing how to express either as a power of the other.

1. Introduction

Consider the following problem:

- Alice knows a solution to the Discrete Log Problem (i.e. for a particular α, β and N, she knows the exponent x such that $\alpha^x \equiv \beta \pmod{N}$ holds).
- Alice wants to convince Bob that she knows x.
- Alice is not willing to reveal any information (in the sense defined in the next section)

---

[*]Partially supported by the Netherlands Organisation for the Advancement of Pure Research (ZWO).
[**]Currently at Facultad de Matemáticas, Universidad Católica de Chile, Casilla 6177 Stgo., Santiago, Chile; partially supported by DIUC Grant #211/86.

about the value of x.

- Bob accepts an exponentially small chance that Alice is cheating, i.e. that she pretends to know an x but doesn't. More precisely, the chance that Alice succeeds in cheating without being detected by Bob, will be 2^{-T}, where T is proportional to the time and space required.

In this paper we present a number of protocols which solves this problem, both for the case N a prime, and for the case $N = P_1P_2$, where P_1 and P_2 are prime and of roughly the same size. Notice that there is no probabilistic polynomial time algorithm known for finding x given α, β and N. But even when Alice is restricted to polynomial computational power (as we will assume), this protocol is of interest, since given α and N she can choose $x \in [1, N-1]$ with $\gcd(x, \phi(N)) = 1$ at random and then compute β simply by exponentiation.

In this paper we define the notion *(almost) no information* which is very similar to "zero knowledge", introduced by Goldwasser, Micali and Rackoff [GMR85] (and which has nothing to do with Shannon-information). The difference is that in the GMR model the prover has unlimited computational power, whereas in our model her power is only polynomial with coin flipping. In section 7 we illustrate the need for such a model by giving an example in which both parties have a symmetrical position, and where it is reasonable to assume that neither has unlimited computational power.

As far as we know, no other protocol with the same functionality has been presented. Very recent results by Goldreich, Micali and Wigderson [GMW86], Brassard and Crepeau [BrCr86], and Chaum [Ch86], however, all imply the following: if Alice has a certificate (or witness) of a particular statement which can be verified in polynomial time, then there exists a polynomial time protocol in which she can convince Bob that she has a certificate, without releasing any knowledge (or information in [Ch86]) about the value of this certificate; consequently, there exists a polynomial time protocol for showing possession of the Discrete Log. Nevertheless, these protocols are not very practical. An important merit of the protocols presented here is their practical feasibility.

The structure of this paper is as follows: The next section describes the model and the notion of information under which we prove our protocols secure. In section 3 and 4 we present the protocols together with their proofs of security in the prime and composite cases, respectively. Section 5 is devoted to a specific variation which surprisingly turns out to be *in*secure. Section 6 gives a protocol to convince another party that two elements generate the same subgroup in Z_N^*. The paper ends with an example and two open problems.

2. The model.

In this paper we will use the model developed in [BKP85], but with some modifications. Below we briefly sketch this model using a modified notation. It should be pointed out that this sketch assumes familiarity with [BKP85] or [GMR85].

We think of a protocol as occurring between two Probabilistic Turing Machines (PTM's) A

and B which operate synchronously. Each PTM has, besides a computation tape and a random tape, a one-way infinite tape for incoming messages. We call this tape the "mailbox" of the machine. The PTM's communicate by writing into each others mailbox. We call each of the machines in such a system a **CPTM**, for Communicating Probabilistic Turing Machine.

Now consider a **system** of two CPTM's A and B. The system $[A;B]$ halts whenever A or B halts; the system accepts only when B halts and accepts. Throughout this paper A will interactively demonstrate possession of a secret to B, so (using the terminology of [GMR85]) we call A the **prover** and B the **verifier**. In [BKP85] this secret is the factorisation of a large composite number. But in general, the solution to an instance of any problem assumed not solvable in random polynomial time may serve as a secret. We define I_A as the pair constisting of the problem instance *and* the secret; I_A is usually created by A, and is considered the input for the system $[A;B]$. Given I_A, we define I_A^* as the single problem instance, thus *without* the secret; we assume that A sends I_A^* to B before the actual protocol starts. For example, in our Discrete Log protocols $I_A = (\alpha, \beta, N, x)$, and $I_A^* = (\alpha, \beta, N)$.

For simplicity, we explicitly force the time ordering in the messages by requiring that A and B alternately write one symbol in the other's mailbox. If a party has nothing to communicate it writes the special null symbol, ∇, not used for any other purpose in the communication. We also assume that both parties do not write superfluous null symbols, so the places where null symbols are written is a function of I_A^*. We define $\Omega := \bigcup_{n=0}^{\infty} \{0, 1, \nabla\}^n$, and the contents of a mailbox as an element of Ω.

The conversation between A and B, defined as the ordered pair containing their respective mailboxes, is considered as the output of the system $[A;B]$. It depends only on the instance I_A and the bits on the random tapes of A and B. This conversation is denoted as $conv([A;B](I_A))$. Then $\Pr(conv([A;B](I_A))=c)$ is defined as the probability that $c \in \Omega^2$ occurs as the conversation between A and B resulting from the initialising instance I_A, under the assumption that the bits on the random tapes of A and B are chosen independently and uniformly.

The following definitions (which are modifications of part (iii) of the definition of an A-simulator preceding theorem 1 in [BKP85]), will serve to make precise the kind of security achieved by our protocols. Informally speaking, they state that the prover A releases no information if there exists a probabilistic polynomial-time simulating machine which, when initialised with I_A^* and for all possible verifiers B', produces simulated conversations between A and B that have (almost) the same probability distribution as the true conversations between A and B. This simulating machine, denoted S_{A^*}, is a PTM S that contains another machine A^*. This A^* is called as a subroutine and outputs a simulation of A's part of the conversation. Input for S_{A^*} is the problem instance without the secret, I_A^*; the output is denoted as $output(S_{A^*}(I_A^*))$.

Definition 1 . The prover A releases **no information** if there exists a polynomial-time (simulating) machine S_{A^*}, such that for all CPTM's B', all initialising instances I_A, and all possible conversations $c \in \Omega^2$,

$$\Pr(conv([A;B'](I_A)) = c) \quad = \quad \Pr(output(S_{A^*}(I_A^*)) = c).$$

The next definition covers the case when the two probability distributions may differ slightly.

Definition 2. The prover A releases **almost no information** if there exist an ϵ, $0 \leqslant \epsilon < 1$, and a polynomial-time (simulating) machine S_{A^*} such that for all CPTM's B', all initialising instances I_A of length l

$$\sum_{c \in \Omega^2} \left| \Pr(conv([A;B'](I_A)) = c) \quad - \quad \Pr(output(S_{A^*}(I_A^*)) = c) \right| \leqslant \epsilon^l.$$

For describing a cryptographic protocol in the model presented, we will use the same protocol notation throughout the paper. The meaning of this notation is straightforward; only the next few things might need explanation:

— Expressions shown on the left or right are known to that party only, and are secret from the other party.

— 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.

— $e \in_R S$ means that an element e is chosen at random from the set S, where all elements of S have an equal probability of being chosen, 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.

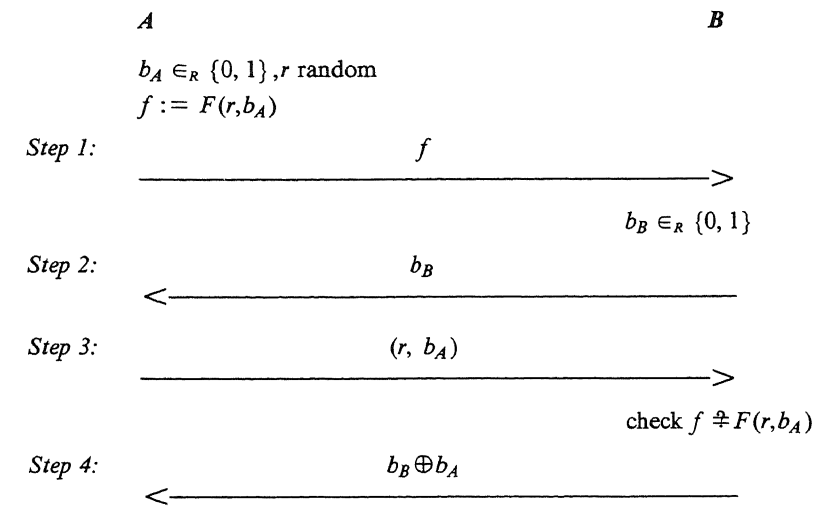
The proofs of security for our protocols are considerably simplified by the fact that there is essentially no two-way communication. The nature of the protocols presented here is such that the bits that B reads from his random tape, can also be generated by a **mutually trusted random source**. The correctness of the protocols lies in the randomness of the bits generated, however, there is no reason for B to hide these bits. If a protocol has this property, we say it is **verifier-passive**.

Several **coin flipping** protocols are widely known which allow A and B to generate mutually trusted random bits, see e.g. [Bl82]. Below we briefly describe the general nature of these protocols. Let $b \in \{0,1\}$ be a bit, r be some random padding, and assume that A and B agree on a function F with the following two properties:

1) given the function F and the value $F(r,b)$, the bit b cannot be computed by B;
2) given the function F and the value $F(r,b)$, a pair (r', b') for which $F(r,b) = F(r',b')$ and $b \neq b'$ cannot be found by A.

Then A and B can use the following protocol, called Γ, for generating mutually trusted random bits:

Protocol Γ: Coin flipping



Note that in this protocol the role of A and B can be interchanged, but this might depend on the model of computation.

For generating bit strings of length T, this protocol can be extended to a protocol Γ_T in two different ways: a sequential version, where Γ is repeated T times, and a parallel version, where both parties send message tuples of length T. We will use this coin-flipping as a sub-protocol.

Because of the time ordering in the conversation, the meaning of each cell in the mailboxes of A and B is completely determined by Π, the kind of protocol used, the security parameter T and the initialising instance I_A. So in the mailboxes we can distinguish between sequences of cells dedicated to the coin-flipping protocol Γ_T, and sequences of cells dedicated to the top-level protocol Π. More formally, if $\vec{b} = (b_1, \ldots, b_T)$ are the bits generated through Γ_T, we define $\pi_A(I_A, \vec{b})$ as those cells written by A (in B's mailbox) for protocol Π only, with null symbols at all other places; similarly, $\gamma_A(I_A, \vec{b})$ is the output of A with regard to Γ_T only, with null symbols at all other places. B's part of the conversation is split similarly in π_B and γ_B. For the simulating machines A^* and S we define π_{A^*}, γ_{A^*}, π_S and γ_S on input I_A^* and $\vec{b}$ in the corresponding way.

Theorem 1. Suppose that the protocol Π is verifier-passive, that a coin-flipping protocol Γ_T is used, and that a CPTM A^* exists such that

$$\forall\ I_A,\ d \in \Omega,\ \vec{b} \in \{0, 1\}^T\colon\ \Pr(\pi_A(I_A, \vec{b}) = d) \ = \ \Pr(\pi_{A^*}(I_A^*, \vec{b}) = d).$$

Then A releases no information through protocol Π.

Proof: We have to show that a polynomial-time simulating machine S_{A^*} can be constructed which simulates the conversation between A and B. This conversation, i.e. the contents of A's

and B's mailboxes, can be split up in π_A, π_B, γ_A and γ_B. By assumption, π_A can be simulated by π_{A^*}. And π_B consists of null symbols only, since Π is verifier-passive; therefore π_B is also simulatable. In general, it is easy for S_{A^*} to simulate Γ_T. However, while simulating Π, machine A^* has to guess in advance the bits $\vec{b}$ resulting from Γ_T, otherwise the simulation fails. In the parallel version of Γ_T the probability of A^* guessing correctly all bits is only 2^{-T}. In the sequential version of Γ_T this probability is ½ in each round. But as soon S_{A^*} realizes that the wrong coin is being simulated, the machine is reset to the state it had when that round was entered and tries that round again. Because of this fact, the error probability can be made arbitrary small. Though S_{A^*}'s expected running time is increased by a factor $2T$ when compared to A, the simulation still runs in probabilistic polynomial time. □

Theorem 2 establishes the analogous result regarding protocols which transfer *almost* no information.

Theorem 2. Suppose the protocol Π is verifier-passive, that a coin-flipping protocol Γ_T is used, and that an ϵ, $0 \leqslant \epsilon < 1$, and a CPTM A^* exists such that for all CPTM's B',

$$\forall\ I_A, \vec{b} \in \{0,1\}^T: \sum_{d \in \Omega} \left| \Pr(\pi_A(I_A, \vec{b}) = d) \ - \ \Pr(\pi_{A^*}(I_A^*, \vec{b}) = d) \right| \leqslant \epsilon^l,$$

where $l := |I_A|$. Then A releases almost no information through the protocol Π.

Proof: The proof is analogous to the proof of Theorem 1. □

Machine A^* in the statements of Theorems 1 and 2 is called a **prover-simulator** machine or just an **A-simulator** machine.

From now on we denote the bits produced by a coin flipping sub-protocol Γ by the word ***COIN FLIPPING***, and a two-sided arrow. Furthermore, Z_N is the additive group (mod N); and Z_N^* is the multiplicative group (mod N).

3. Protocols for proving possession of the discrete logarithm modulo a prime number.

The problem is the following : A knows a solution to the equation $\alpha^x \equiv \beta \ (\text{mod } P)$, where we assume that x is randomly chosen from $[1, P-1]$. P, α, β are public and B wants to be convinced that A knows x. A wants to convince B, but does not want to release any information about x.

We will give two protocols for this problem. Our first example is an easier protocol; however, it works only if α and β both generate the same sub-group in Z_P^* and A is willing to acknowledge this. If α and β do not generate the same sub-group, protocol 1 releases information about the index of $\langle\beta\rangle$ in $\langle\alpha\rangle$. An intuitive way to think about this protocol is to consider the expression $h_i := \alpha^{e_i}$ made public as lying somewhere between α and β; upon getting the value of the bits, A shows either how to express h_i as a power of α, or how to express β as a power of h_i.

Protocol 1: $\alpha^x \equiv \beta \pmod P$; P, α, β public; x with $\gcd(x, P-1) = 1$ a secret of A; α, β generate the same sub-group.

	A		B
	$e_1, \ldots, e_T \in_R Z^*_{P-1}$		
	$h_i := \alpha^{e_i} \pmod P$		
Step 1:		$h_1, \ldots, h_T$ ——>	
Step 2:		<—— *COIN FLIPPING:* $b_1, \ldots, b_T \in_R \{0,1\}$ ——>	
	if $b_i = 0$ then $s_i := e_i$		
	if $b_i = 1$ then $s_i := xe_k^{-1} \pmod{P-1}$		
Step 3:		$s_1, \ldots, s_T$ ——>	
			if $b_i = 0$ then check $\alpha^{s_i} \stackrel{?}{=} h_i$
			if $b_i = 1$ then check $h_i^{s_i} \stackrel{?}{=} \beta$

Theorem 3.
(a) A can cheat in protocol 1 with probability at most 2^{-T} if she does not know x, and
(b) there exists a polynomial-time prover simulator A^*.

Proof:
(a) Correctness: If A does not know x, then she is not able to compute both possible exponents to be released in step 3. Hence she will get caught with probability at least ½ with each h_i. Thus A will get caught cheating with probability at least $1 - 2^{-T}$.
(b) Security: We exhibit a simulator A^* which, for random bits $b_1, \ldots, b_T$, produces random $h_1, \ldots, h_T$ in Z^*_P, along with r_i such that $\alpha^{r_i} \equiv h_i$ if $b_i = 0$ and s_i such that $h_i^{s_i} \equiv \beta \pmod P$ if $b_i = 1$. We construct A^* as follows:

***A*-Simulator for protocol 1:**

1: $b_1, \ldots, b_T \in_R \{0,1\}$

2: If $b_i = 0$ then $s_i \in_R Z^*_{P-1}$ and $h_j := \alpha^{s_j} \pmod P$.

3: If $b_i = 1$ then $s_i \in_R Z^*_{P-1}$ and $h_i := \beta^{\sigma_i}$, where $\sigma_i \equiv s_i^{-1} \pmod{P-1}$.

4: Output h_i, b_i and s_i for $i \in \{1, \ldots, T\}$.

The reader can verify that the b_i's, h_i's, r_i's and s_i's produced by simulator A^* have the same joint probability distribution as the corresponding numbers produced by A in an execution of protocol 1. Note that the computations in step 3 can be done in polynomial time using Euclid's algorithm. By Theorem 1, protocol 1 reveals *no information*. □

Protocol 1 has the advantage that it can be performed sequentially by using $T = 1$ and repeating step 1 to step 3 many times. Protocol 2 below can be proved to be secure only when it is performed in parallel; however, it can also be used if α and β do *not* generate the same group.

Protocol 2: $\alpha^x \equiv \beta \pmod P$; P, α, β public; x a secret of A.

	A		B
	$e_i \in_R Z_{P-1}$		
	$h_i := \alpha^{e_i} \pmod P$		
Step 1:		$h_1, \ldots, h_T$ ——>	
Step 2:		*COIN FLIPPING:* $b_1, \ldots, b_T \in_R \{0, 1\}$ <——	
	if $b_i = 0$ then $s_i := e_i$		
	if $b_i = 1$ then $s_i := e_i - e_j \pmod{P-1}$, with $j := \min\{i : b_i = 1\}$		
Step 3:		$s_1, \ldots, s_T$ ——>	
			if $b_i = 0$ then check $\alpha^{s_i} \stackrel{?}{=} h_i$
			if $b_i = 1$ then check $\alpha^{s_i} \stackrel{?}{=} h_i h_j^{-1}$
	$\xi := x - e_j \pmod{P-1}$		
Step 4:		ξ ——>	
			check $\alpha^{\xi} \stackrel{?}{=} \beta h_j^{-1}$

Theorem 4.
(a) A can cheat in protocol 2 with probability at most 2^{-T} if she does not know x, and
(b) there exists a polynomial-time prover simulator A^*.

Proof:
(a) Correctness: Suppose that A does not know x. Then she will get caught with probability at least ½ for each h_i, for $i \neq j$. This is because A can never answer both possible cases to be sent in step 2. Now, independent of what j is, A's chance of being caught with h_j is also at least ½, because she cannot know e_j *and* pass the check after step 4. So the only way A can pass all the checks in step 3 and step 4 is by guessing correctly what the vector $\vec{b}$ will be. This happens with probability $1 - 2^{-T}$.

(b) Security: Note that protocol 2 is verifier-passive. We exhibit a simulator machine A^* which produces messages and random bits which have the same joint probability distribution as messages from A and mutually trusted random bits in an execution of the protocol. By Theorem 1 it then follows that protocol 2 releases no information.

In the remainder of the proof we have $K := \{i : b_i = 0\}$, $L := \{i : b_i = 1\}$, $k \in K$ and $l \in L$. For

random bits $b_1, \ldots, b_T$, A^* must produce random $h_1, \ldots, h_T$ in Z_P^*, along with e_k such that $\alpha^{e_k} \equiv h_k \pmod{P}$ for each $k \in K$. For each $l \in L$, simulator A^* must produce the difference $s_l \equiv e_l - e_j \pmod{P-1}$ satisfying $\alpha^{s_l} \equiv h_l h_j^{-1} \pmod{P}$, where $j = \min\{i : i \in L\}$. Finally, A^* must produce the difference $\xi \equiv x - e_j \pmod{P-1}$ satisfying $\alpha^{\xi} \equiv \beta h_j^{-1} \pmod{P}$. We construct machine A^* as follows:

A-Simulator for protocol 2:

1: $b_1, \ldots, b_T \in_R \{0,1\}$. Let K and L be defined as before.

2: For $k \in K$ choose $s_k \in_R Z_{P-1}$, and let $h_k := \alpha^{s_k} \pmod{P}$.

3: Choose $\xi \in_R [1, P-1]$. For $l \in L$ choose $s_l \in_R [1, P-1]$.
For $l \in L - \{j\}$ let $h_l := \alpha^{s_l - \xi}\beta$.

4: Let $h_j := \alpha^{-\xi}\beta$.

5: Output h_i, $b_i$and s_i for $i = 1, \ldots, T$, and ξ.

Observe for step 4 that $h_l \equiv \alpha^{s_l - \xi}\beta \equiv \alpha^{s_l - \xi + x} \equiv \alpha^{(e_l - e_j) - (x - e_j) + x} \equiv \alpha^{e_l}$ and for step 5 that $h_j \equiv \alpha^{-\xi}\beta \equiv \alpha^{-(x-e_j)+x} \equiv \alpha^{e_j}$. Now it follows immediately that the b_i, the e_i, the h_k, the s_l and the ξ produced by A^* have the same joint probability distribution as the ones produced by A in an execution of protocol 2. □

The crucial difference between A and A^* is, that the simulator A^* does not know the actual values of the e_l (because it does not know x), but only their differences $(\bmod P-1)$. Since the protocol does not reveal the actual values of e_l and x, but only their difference with e_j taken $(\bmod P-1)$, the protocol is secure.

4. A protocol for proving possession of a discrete logarithm modulo a composite public key.

In this section we consider the analogues of protocols 1 and 2 modulo *composite* numbers, where we assume that the proving party A knows the factorization of N (henceforward called N_A).

So the problem is the following: A knows a solution to the equation $\alpha^x \equiv \beta \pmod{N_A}$, where N_A is a composite modulus whose factorisation is known to A only. Again α, β are public and B wants to be convinced that A knows x. And A wants to convince B, but does not want to release any information about either x or the factorization of N_A. Note that the operations on the numbers themselves are carried out modulo N_A, but on the exponents modulo $\phi(N_A)$.

First consider the analogue of protocol 1. As is easily verified, the protocol is feasible, but cannot be proven to be secure. The crucial point here is that when we look at the simulator for protocol 1, this simulator cannot execute step 3 since it does not know $\phi(N_A)$ (namely, this is essentially equivalent to knowing the factorisation of N_A). The simulator must generate pairs (h,s) for which $h^s \equiv \beta \pmod{N_A}$. But since we cannot prove that any simulator can do this in

polynomial time (in fact this does not seem very likely), this modification of protocol 1 cannot be proven secure.

As we will show now, protocol 2 can be used for composite numbers as well, be it at the cost of a very small probability of insecurity.

Protocol 3: $\alpha^x \equiv \beta \pmod{N_A}$; N_A, α, β are public; x and the factorization of N_A a secret of A.

This protocol is exactly the same as protocol 2 except that exponents are chosen modulo $\phi(N_A)$. The sums and differences of exponents are also revealed modulo $\phi(N_A)$.

Theorem 5 . *(a)* A can cheat in protocol 3 with probability 2^{-T} if she does not know x, and *(b)* there exists a polynomial-time prover simulator A^* that produces a conversation with *almost* the same probability distribution.

Proof:

(a) Correctness: the same as for protocol 2.

(b) Security: The added complication is that A must not only know x, but, in order to perform the protocol, must also know the value of $\phi(N_A)$. Hence the possibility arises that A may release some information about the factorization of N_A. However, we can use the same simulator machine A^* as in the proof of security for protocol 2, except that A^* chooses exponents uniformly from the set $\{1,..., N_A\}$. We can do this since the construction for A^* involves no exponent arithmetic modulo $\phi(N_A)$. Thus the value of $\phi(N_A)$ is not used by A^*. The resulting probability distribution is not identical to the one generated by A, who choses her exponents in $[1,..., \phi(N)]$. However, a straightforward computation shows that the difference of these distributions, as expressed by the sum of absolute differences in definition 2, is negledgibly small. Use here that $\frac{N_A - \phi(N_A)}{N_A}$ is exponentially small in the size of N_A (assuming that N_A is the product of two prime numbers of nearly the same size). Thus protocol 3 releases *almost no information.* □

5. An *in*secure protocol for proving possession of a discrete logarithm modulo a composite number when the factorization of the number is not known.

The problem is the following: A knows a solution to the equation $\alpha^x \equiv \beta \pmod{N}$, where N is a composite modulus. α, β are public and B wants to be convinced that A knows x. A wants to convince B, but does not want to release any information about x. Here A does *not* know the factorization of N.

Protocol 4: $\alpha^x \equiv \beta \pmod{N}$; N, α, β are public; x a secret of A; A does not know a factor of N.

This protocol is the same as protocol 3 except that exponents are randomly chosen between 1 and N. Sums and differences of exponents are *not* reduced modulo $\phi(N)$ (since A does not know $\phi(N)$) and are instead released as integer sums.

Theorem 6. Protocol 4 releases information with regard to definition 2.

Proof: As before, $K := \{i : b_i = 0\}$, $L := \{i : b_i = 1\}$, $k \in K$ and $l \in L$. Define $s_{\max} := \max\{s_l : l \in L\}$, and $s_{\min}$ similarly. Because we treat the s_l and the e_l as integers, there is no wrap around modulo $\phi(N)$. So $s_{\max} - s_{\min} = (s_{\max} + e_j) - (s_{\min} + e_j) = e_{\max} - e_{\min} =: \lambda$; note that λ is the length of the smallest interval containing all e_l, for $l \in L$. Because $e_{\min} \in [1,N]$ and $e_{\max} = e_{\min} + \lambda \in [1,N]$ we find that $e_{\min} \in [1, N-\lambda]$. This implies that $x = e_{\min} + (x - e_j) - (e_{\min} - e_j) = e_{\min} + \xi - s_{\min} \in [1 + \xi - s_{\min}, N - \lambda + \xi - s_{\min}]$. Now it is immediately clear that A^*, who does not know x, cannot produce a conversation with the same probability distribution as A. This proves that protocol 4 leaks information in the sense of definition 2. □

We consider the *Shannon* information released by protocol 4. When λ is very close to N, then the number of possible values for x drops from N to $N - \lambda$. It is easy to see that when the number of equally likely possibilities reduces with a factor 2^{-m}, then m bits of Shannon information are revealed. This (or a similar computation using entropy) shows that the amount of information released by this protocol equals $\log_2(N/(N-\lambda)) = \log_2(1-\lambda/N)^{-1}$. Let Λ denote the stochastic variable for the length of the interval, and let $Pr(\Lambda = \lambda)$ be the probability that λ is the length of the interval. Then we define the average release of information as $\sum_{\lambda=1}^{N} Pr(\Lambda=\lambda) \log_2 \frac{1}{(1-\lambda/N)}$. A straightforward computation of this sum shows that the average release of information for this protocol is approximately $\log_2 |L| \leqslant \log_2 T$, where $|L|$ is the cardinality of L.

6. A protocol for proving that two elements generate the same group in Z_P^* or Z_N^*.

Let $\alpha \in Z_P^*$ or Z_N^*, and let $\langle\alpha\rangle$ denote the multiplicative subgroup generated by α. Protocol 1, 2 or 3 can all be used to show that $\langle\alpha\rangle = \langle\beta\rangle$ provided that A knows a relation between α and β. Note that proving that $\langle\alpha\rangle = \langle\beta\rangle$ is a problem not known to be solvable in polynomial time even modulo a prime number.

Protocol 5: $\langle\alpha\rangle = \langle\beta\rangle$ in Z_N^*; α, β and N are public; x for which $\alpha^x \equiv \beta \pmod N$ is a secret of A.

> Use protocol 1,2 or 3 in both directions: A shows to B that she knows how to express β as a power of α and how to express α as a power of β.

The correctness of this protocol is easy to understand: $\langle\alpha\rangle = \langle\beta\rangle$ if and only if there exists an x such that $\alpha^x \equiv \beta \pmod N$ and a y such that $\beta^y \equiv \alpha \pmod N$. A knows x and $\phi(N)$, thus A can compute $y \equiv x^{-1} \pmod{\phi(N)}$ in polynomial time. So A can perform protocol 1, 2 or 3 in both directions, thus showing knowledge of both x and y. The security of sequential use for these protocols lies in the kind of security proved. In the terminology of Berger, Kannan, Peralta [BKP85], the proofs of security of protocols 1-3 show that these protocols are **strongly secure** and

therefore can be executed one after the other without loss of security.

7. Applications and open problems.

We conclude with an application and two open problems:

1. Diffie-Hellman Key Exchange can be made more secure:

In a fair execution of the so-called Diffie-Hellman Key Exchange protocol [DiHe76] both parties choose an exponent x_A and x_B, they send $\beta_A := \alpha^{x_A} \pmod P$ and $\beta_B := \alpha^{x_B} \pmod P$ to each other, and they use $\beta_B^{x_A} = \beta_A^{x_B}$ as their secret communication key.

Now suppose B has two polynomial time algorithms F_1 and F_2. On input α, P, β_A algorithm F_1 yields δ, which B uses as his β_B. Using δ^{x_A} as a key, A sends a message to B. This message, together with α, P, β_A, is fed to algorithm F_2 which has x_A as output.

As far as we know it has not been proven that such algorithms F_1, F_2 and such δ do not exist. This would be undesirable, because when B knows x_A he can pretend to be A to a third party. Using protocol 1 or 2 in this paper we can extend the Diffie-Hellman Key Exchange protocol by requiring both parties to show they know the discrete logarithm x. Then for B's attack to work, he would have to know $\log_\alpha \delta$ besides δ. But as is easily verified, this implies that B has a polynomial algorithm for the Discrete Log Problem (he can himself simulate the message sent by A).

2. Can the same protocols be used with two generators?

Suppose that A wants to prove she knows x_1 and x_2 such that $\alpha_1^{x_1}\alpha_2^{x_2} \equiv \beta \pmod N$. Note that the status of this "Relaxed" Discrete Log Problem is not clear.

3. How hard is it to find any root of a given number β modulo a composite N of which the factorization is not known?

This is the problem mentioned at the end of section 3: can one find in polynomial time a pair (h, s), $s > 1$, which is a solution for $h^s \equiv \beta \pmod N$, or is this problem reducible to a hard problem?

Acknowledgements

We are grateful to Oded Goldreich for his critical and helpful comments on an earlier version. Also we would like to thank Bert den Boer for his help and remarks.