Cryptanalysis of an Efficient Proof of Knowledge of Discrete Logarithm

Sébastien Kunz-Jacques[1,2], Gwenaëlle Martinet[1], Guillaume Poupard[1], and Jacques Stern[2]

[1] DCSSI Crypto Lab, 51 boulevard de La Tour-Maubourg
F-75700 Paris 07 SP, France
{Sebastien.Kunz-Jacques,Gwenaelle.Martinet,Guillaume.Poupard}@sgdn.pm.gouv.fr
[2] École normale supérieure, Département d'informatique
45 rue d'Ulm, F-75230 Paris Cedex 05, France
Jacques.Stern@ens.fr

Abstract. At PKC 2005, Bangerter, Camenisch and Maurer proposed an efficient protocol to prove knowledge of discrete logarithms in groups of unknown order. We describe an attack that enables the verifier to recover the full secret with essentially no computing power beyond what is required to run the protocol and after only a few iterations of it. We also describe variants of the attack that apply when some additional simple checks are performed by the prover.

Keywords: Public key cryptanalysis, discrete logarithm, proof of knowledge.

1 Introduction

Since the seminal paper of Diffie and Hellman [10], the discrete logarithm problem has been considered a fundamental stone of public key cryptography. In order to define this problem in a general setting, we consider a multiplicative group $\mathcal{G}$ and an element $g \in \mathcal{G}$. We note ω the multiplicative order of g in $\mathcal{G}$ i.e. the smallest non-zero positive integer ω such that $g^\omega = 1$. The set $\langle g \rangle = \left\{g^i\right\}_{i\in\mathbb{Z}}$ of powers of g is a subgroup of $\mathcal{G}$ with ω elements. For any member $y \in \langle g \rangle$, there exists a unique integer $x \in \{0, ...\omega - 1\}$ such that $y = g^x$; by definition x is the discrete logarithm of y in base g. The computation of such discrete logarithms is considered to be intractable in many groups of cryptographic interest such as modular groups or elliptic curves.

An interesting question is how to prove knowledge of a discrete logarithm of a public data without revealing any other information about this value. Such a problem is closely related to the concept of zero-knowledge introduced in 1985 by Goldwasser, Micali and Rackoff [13]. A well-known and very nice solution was proposed by Schnorr [17] in 1989. In this two party-protocol, a prover who knows the discrete logarithm x of a public value y interacts with a verifier; if the prover is able to correctly answer the verifier's challenges, he proves knowledge of x. Two complementary security aspects can be analyzed; firstly, the soundness property shows that if a prover is able to correctly answer the challenges then he must know the secret x. This proof is based on the notion of knowledge extractor that can extract the secret from the prover using rewinding techniques. Secondly, the zero-knowledge property shows that the execution of the protocol does not leak any information about the secret x, even if the verifier tries to bias its challenges. The proof is based on the notion of simulation of the communications.

In the Schnorr scheme, the soundness property can be easily proved since the secret is immediately derived from two correct and distinct answers corresponding to the same "commitment" sent by the prover as its first message. Deciding if the protocol is zero-knowledge is still an open problem when large challenges are used and if they are not randomly chosen by the verifier. It is significant to note that the proof of soundness strongly relies on the knowledge of the order ω of the basis g. Surprisingly, if this order is not known, for example is the context or RSA groups, the basic extraction strategy no longer applies. It is still possible to prove the security of the scheme used as an identification scheme [12, 16] but, in groups of unknown order, Schnorr based proofs cannot be considered as proofs of knowledge. This interesting open problem has attracted the interest of several research papers [11, 9] and, at PKC 2005, Bangerter, Camenisch and Maurer [1] proposed an efficient protocol, the so-called Σ^+-Protocol to prove knowledge of discrete logarithms in groups of unknown order. This scheme is derived from the Σ-Protocol whose paternity is unclear. The name was first proposed in 1997 by Cramer [7] in his PhD thesis and used by Cramer and Damgård [8] but original ideas can be found in the Schnorr scheme [17] and even previously in [5, 4, 2]. However, Girault [12] was the first to observe, in 1991, that the knowledge of the underlying group order was not necessary to carry Schnorr's like proofs.

In this paper, we show that the proposal in [1] is not secure since a dishonest verifier can obtain the secret of the prover. The main flaw in [1] is that the authors assume that some parameters needed for a protocol run are honestly chosen by the verifier; in the Σ^+ protocol, the prover never checks, and is not able to check, that these parameters actually have the correct form. Our attack takes advantage of this mistake. Thus, even if the protocol if proved in [1] to be a zero-knowledge proof of knowledge, the assumptions made in the proof cannot be verified with the described protocol. To fulfil the proof's assumptions, some additional and non obvious checks are needed which may drastically reduce the protocol efficiency. Some other solutions may be considered but they require to revise the protocol's proof.

Notations and organization of the paper. Throughout this paper, we use the following notation: for any integer n,

- $\mathbb{Z}_n$ is the set of integers modulo n,
- ${\mathbb{Z}_n}^*$ is the multiplicative group of invertible elements of $\mathbb{Z}_n$,
- $\varphi(n)$ is the Euler totient function, i.e. the cardinality of ${\mathbb{Z}_n}^*$,
- $\mathrm{ord}(g)$ is the order of an element $g \in {\mathbb{Z}_n}^*$,
- $\lambda(n)$ is the Carmichael's lambda function defined as the largest order of the elements of ${\mathbb{Z}_n}^*$.

It is well known that if the prime factorization of an odd integer n is $\prod_{i=1}^{\eta} {q_i}^{f_i}$ then $\varphi(n) = \prod_{i=1}^{\eta} {q_i}^{f_i-1}(q_i - 1)$ and $\lambda(n) = \mathrm{lcm}_{i=1\dots\eta}\left({q_i}^{f_i-1}(q_i - 1)\right)$.

The paper is organized as follows: section 2 recalls the Σ^+-protocol [1]. Then, in section 3, we make some security related observations which lead to a practical cheating strategy. We also observe in this section and in section 4 that several simple and natural countermeasures do not succeed into defeating our strategy. Finally, annex A gives a detailed analysis of the attack complexity and annex B describes a detailed algorithm of independent interest, strongly inspired of the Pohlig-Hellman algorithm [14], to compute discrete logarithms in our setting.

2 The Σ^+-Protocol

Let us now briefly recall the Σ^+-protocol using the notations of [1]. Let H be an arbitrary group whose order needs not to be known. For example, H can be the set ${\mathbb{Z}_n}^*$ for a composite RSA modulus n. Let h be an element of H such that the computation of discrete logarithms in base h is intractable.

The Σ^+-protocol is a proof of knowledge of discrete logarithms of elements in H, in base h. Roughly speaking, this means that, for a given $y \in H$, a prover can convince a verifier that he knows an integer x such that $y = h^x$. As we will see in the rest of this paper, this protocol is not a zero-knowledge proof of knowledge of discrete logarithm since the prover reveals some information about his secret x when interacting with a dishonest verifier.

The proof requires a generator $\mathcal{D}_{\mathcal{S}}(k)$ that outputs a pair (n, g) s.t. n is an RSA modulus, $g \in {\mathbb{Z}_n}^*$ and it is hard to compute $u \in {\mathbb{Z}_n}^*$ and an integer $e > 1$ fulfilling $u^e = g \bmod n$. It is stated in [1] that "[the authors] *assume that* $n = (2p+1)(2q+1)$ *with* p, q, $(2p+1)$ *and* $(2q+1)$ *being primes, and that* $g \in QR_n$*, where* QR_n *is the subgroup of quadratic residues of* ${\mathbb{Z}_n}^*$". However even if this assumption appears to be used in the security analysis, at least in a side remark to prove the statistical zero-knowledge property of the protocol, it is not guaranteed by the protocol itself.

We still need a few additional notations coming from [1]:

- k is a security parameter,
- $a \in_U A$ means that the element a is randomly chosen in the set A using a uniform distribution,
- the equality symbol $\doteq$ is used to denote definitions,
- the secret exponent x is in the range $[-\Delta x, \Delta x]$ and the related public element of H is $y = h^x$,
- l_z is an integer parameter related to the security parameter k,
- c^+ is another parameter that determines the set $\{0, \dots, c^+\}$ in which the verifier picks its challenges c,
- $\mathrm{commit}(\gamma, r)$ is a computationally binding and statistically hiding commitment scheme that commits γ using the random value r; to open the commitment one reveals γ and r.

Prover		Verifier
Private input : x in $[-\Delta x, \Delta x]$		
	Common input : h and $y = h^x$ both in H	
		$(n, g) \leftarrow \mathcal{D}_{\mathcal{S}}(k)$ $\rho \in_U [0, 2^k \lfloor n/4 \rfloor]$ $g_1 \doteq g^\rho \bmod n$
	$\xleftarrow{(g_1, g, n)}$	
$\mathsf{x} \in_U [0, \lfloor n/4 \rfloor]$ $\mathsf{y} \doteq g_1^x g^{\mathsf{x}} \bmod n$ $r \in_U [-2^{l_z} c^+ \Delta x, 2^{l_z} c^+ \Delta x]$ $t \doteq h^r$ $\mathsf{r} \in_U [-2^{l_z} c^+ \lfloor n/4 \rfloor, 2^{l_z} c^+ \lfloor n/4 \rfloor]$ $\mathsf{t} \doteq g_1^r g^{\mathsf{r}}$ Choose r_{y}; $\overline{\mathsf{y}} \doteq \mathrm{commit}(\mathsf{y}, r_{\mathsf{y}})$ Choose r_{t}; $\overline{\mathsf{t}} \doteq \mathrm{commit}(\mathsf{t}, r_{\mathsf{t}})$		
	$\xrightarrow{(\overline{\mathsf{y}}, \overline{\mathsf{t}}, t)}$	
		$c \in_U \{0, \ldots, c^+\}$
	$\xleftarrow{c}$	
$s \doteq r + cx$ $\mathsf{s} \doteq \mathsf{r} + c\mathsf{x}$		
	$\xrightarrow{(s, \mathsf{s})}$	
	$\xleftarrow{\rho}$	
If $g_1 \neq g^\rho \bmod n$, then halt.		
	$\xrightarrow{((\mathsf{t}, r_{\mathsf{t}}), (\mathsf{y}, r_{\mathsf{y}}))}$	
		If the equalities $\overline{\mathsf{y}} = \mathrm{commit}(\mathsf{y}, r_{\mathsf{y}})$ $\overline{\mathsf{t}} = \mathrm{commit}(\mathsf{t}, r_{\mathsf{t}})$ $h^s = t y^c$ $g_1^s g^{\mathsf{s}} = \mathsf{t}\mathsf{y}^c \bmod n$ hold, then output 1 else output 0

Fig. 1. The Σ^+-Protocol from [1]

The typographic convention of [1] is to use sans serif font for elements related to computations in ${\mathbb{Z}_n}^*$ and standard italic font when dealing with elements of H.

The Σ^+-protocol described in figure 1 performs a kind of parallel proof of knowledge of discrete logarithms in two mathematical structures, H and ${\mathbb{Z}_n}^*$, in a way similar to proofs of equality of discrete logarithms. However, the main original part is that the second structure is not a parameter of the system but is chosen by the verifier and changes from one proof to another.

3 Some security related observations

3.1 A preliminary observation

A first simple security related observation is that some basic checks should be added to the scheme, exactly as for the original Σ-Protocol. This may be considered implicit but it is probably better to make checks explicit in order to avoid dramatic consequences in practical implementations.

More precisely, a remark made by D. Bleichenbacher about the GPS identification scheme during the NESSIE selection process [6] is relevant to the present context; consider a cheating verifier that does not choose the challenge c uniformly in the range $[0, c^+]$ but sends a value much larger than c^+. If the prover does not check that $c \in [0, c^+]$, he reveals $s = r + cx$ with $r \in [-2^{l_z} c^+ \Delta x, 2^{l_z} c^+ \Delta x]$. Then, $s/c = x + r/c$ and, if $c > 2^{l_z+1} c^+ \Delta x$, the verifier obtains $s/c - 1/2 < x < s/c + 1/2$ and consequently the secret $x = \lfloor s/c + 1/2 \rfloor$.

As a consequence, a check on the range of c must be performed by the prover. In the same vein, even if the consequences are not so important, the verifier should also check that the answers s and s lie in consistent ranges; this may be important to perform a full security proof.

This preliminary observation is not used in the sequel and we consider that the order of magnitude of any transmitted data is always checked.

3.2 First observation: n can be chosen in such a way that discrete logarithms in $\mathbb{Z}_n{}^*$ can be efficiently computed

The first immediate idea to attack the Σ^+-Protocol is to make the verifier choose a group $\mathbb{Z}_n{}^*$ in which he can efficiently compute discrete logarithms. For example, such a computation can be made if the Pohlig-Hellman algorithm [14] can be applied efficiently, *i.e.* if the multiplicative order of g is the product of only small prime integers. This situation occurs if n is computed as the product of two primes p and q s.t. $p-1$ and $q-1$ are "smooth", *i.e.* are equal to the product of only small prime factors.

Note that this kind of attack was somewhat considered by the authors of [1] since, as we already mentioned, they explicitly restricted themselves to the opposite situation where p and q are strong primes *i.e.* $(p-1)/2$ and $(q-1)/2$ are also primes. But, even if such a choice seems to be specified for a honest verifier in order to protect him against dishonest provers, a dishonest verifier can choose different kind of parameters to try to attack a honest prover. Such a cheating strategy does not seem to be taken into account since the prover does not try to detect it. The situation is even worse since the prover does not have enough information to check the correctness of n as a product of two unknown strong primes. In [3], Camenisch and Michels have shown how to prove that a modulus is the product of two safe primes. Adding such a proof in Σ^+ would drastically reduce the claimed efficiency of the protocol and render it totally unpractical.

The consequence of this first observation is that a cheating verifier can choose the modulus n s.t. he can further compute easily the following information:

1. $x\rho + \mathsf{x} \bmod \mathrm{ord}(g) \quad (= \log_g(\mathsf{y}))$
2. $r\rho + \mathsf{r} \bmod \mathrm{ord}(g) \quad (= \log_g(\mathsf{t}))$

Furthermore, he obtains from the regular execution of the protocol the answers s and s:

3. $s = xc + r$
4. $\mathsf{s} = \mathsf{x}c + \mathsf{r}$

However, even if we obtain four equations with four unknowns (x, x, r and r), this system cannot be solved to recover the secret x since the equations are not independent. Some more work is therefore needed.

3.3 Second observation: some information may be revealed by a honest prover

If a dishonest verifier chooses the prime numbers p and q s.t. $(p-1)/2$ and $(q-1)/2$ are relatively prime, we know that the maximal order of an element in $\mathbb{Z}_n{}^*$ is given by the Carmichael lambda function $\lambda(n) = \mathrm{lcm}(p-1, q-1) = (p-1)(q-1)/2$. The verifier can choose an element g with such a maximal order which is close to $n/2$. In this case, g is not a quadratic residue in $\mathbb{Z}_n{}^*$.

Then, an idea is to choose $\rho = 1$ in combination with a group $\mathbb{Z}_n{}^*$ where the verifier can compute discrete logarithms. The consequence is that the attacker learns $\log_g(\mathsf{y}) = (x + \mathsf{x}) \bmod \mathrm{ord}(g)$ which can be seen as the secret $x \bmod \mathrm{ord}(g)$ masked with x randomly chosen in the range $[0, \lfloor n/4 \rfloor]$. Since $\mathrm{ord}(g) \approx n/2$, the mask x does not fully hide the value of $x \bmod \mathrm{ord}(g)$ and, from an information theoretic point of view, one bit of information is revealed if x is uniformly distributed modulo $\mathrm{ord}(g)$.

It is quite plausible that by repeating this approach one can deduce the exact value of the secret x from this partial information. However, we propose an additional trick to make the attack straightforward and effective.

3.4 Third observation: parameter ρ can be chosen in such a way that the multiplicative order of g_1 is small

Using both previously exposed ideas, let us consider that the verifier chooses n and g s.t.

- p and q are prime integers,
- $(p-1)/2$ and $(q-1)/2$ are relatively prime,
- $p-1$ and $q-1$ are smooth,
- g is an element of $\mathbb{Z}_n{}^*$ of maximal order $\lambda(n) = (p-1)(q-1)/2$.

Let us now choose $\rho = \lambda(n)/2$. As a consequence, $g_1 = g^\rho = g^{\lambda(n)/2} \bmod n$ has multiplicative order 2.

As explained previously, a cheating verifier is able to compute discrete logarithms and thus obtains from a regular proof

$$\begin{aligned}\log_g(\mathsf{y}) &= x\rho + \mathsf{x} \bmod \mathrm{ord}(g)\\ &= \left(x \times \frac{\lambda(n)}{2}\right) + \mathsf{x} \bmod \lambda(n)\\ &= (x \bmod 2) \times \frac{\lambda(n)}{2} + \mathsf{x} \bmod \lambda(n)\end{aligned}$$

As a consequence, since the mask x is chosen in a range of size approximately $\lambda(n)/2$, the observation of the most significant bit of $\log_g(\mathsf{y})$ reveals the least significant bit of x, *i.e.* the value $x \bmod 2$.

Indeed, if $x \bmod 2 = 0$, then $\log_g(\mathsf{y}) = \mathsf{x}$ is uniformly distributed in the range $[0, \lfloor \frac{n}{4} \rfloor]$. If $x \bmod 2 = 1$, then $\log_g(\mathsf{y}) = \mathsf{x} + \frac{\lambda(n)}{2}$ is now uniformly distributed in $[\frac{\lambda(n)}{2}, \frac{\lambda(n)}{2} + \lfloor \frac{n}{4} \rfloor]$. These intervals are not disjoint but their intersection contains approximately only $\frac{p+q}{4}$ points. Thus, with overwhelming probability, the least significant bit of x leaks from a single execution of the protocol with such a cheating verifier.

In short, we have seen that a dishonest verifier can choose special parameters n, g and ρ in such a way that he can learn the secret x modulo two. Note that this is not detected by a prover who follows the protocol.

Then, the next bits of x can also be obtained by extending this attack. Suppose the verifier knows the k least significant bits $x_0, \ldots, x_{k-1}$ of x, where $x = \sum_{i=0}^{\ell} x_i 2^i$. He then tries to infer the bit x_k. To this end, he chooses the parameters n and g as before with the extra condition that 2^{k+1} divides $\lambda(n)$ and $\rho = \lambda(n)/2^{k+1}$. From the prover's answers during the protocol, he computes $\log_g(\mathsf{y}) = \mathsf{x} + x \times \lambda(n)/2^{k+1} \bmod \lambda(n)$ and considers the value

$$\begin{aligned}\log_g(\mathsf{y}) - \sum_{i=0}^{k-1} x_i 2^i \times \frac{\lambda(n)}{2^{k+1}} &= \mathsf{x} + \sum_{i=k}^{\ell} x_i 2^i \times \frac{\lambda(n)}{2^{k+1}}\\ &= \mathsf{x} + x_k \times \frac{\lambda(n)}{2} + \sum_{i=k+1}^{\ell} x_i 2^{i-(k+1)} \times \lambda(n)\\ &= \mathsf{x} + x_k \times \frac{\lambda(n)}{2} \bmod \lambda(n)\end{aligned}$$

which is either in the range $[0, \lfloor \frac{n}{4} \rfloor]$ or in the range $[\frac{\lambda(n)}{2}, \frac{\lambda(n)}{2} + \lfloor \frac{n}{4} \rfloor]$ according to the value of the bit x_k. As before, the verifier can deduce x_k with very high probability from a single execution of the protocol. The precise algorithm is given in figure 2. In this description, for clarity, the commitment of the values y, t and t are not described. This does not change anything in the attack.

A strategy for breaking the protocol is thus to choose a special value for n, *i.e.* a modulus computed as the product of two primes p and q with smooth values $p-1$ and $q-1$, and a generator g which is of maximal order and thus not a quadratic residue in ${\mathbb{Z}_n}^*$.

- Inputs: the bits $x_0, x_1, \ldots, x_{k-1}$ of x
- Output: the bit x_k of x

1. Generate $n = p \times q$, with p and q prime, $(p-1)/2$ and $(q-1)/2$ relatively primes, $p-1$ and $q-1$ smooth and $p-1$ is divisible by 2^{k+1};
2. Generate $g \in \mathbb{Z}_n{}^*$ of order $\lambda(n) = (p-1)(q-1)/2$;
3. Set $\rho = \lambda(n)/2^{k+1}$ and compute $g_1 = g^\rho \bmod n$;
4. Execute a protocol with the prover:
   (a) Send (n, g, g_1) to the prover;
   (b) Receive $\mathsf{y} = g_1^x g^{\mathsf{x}} \bmod n$, $\mathsf{t} = g_1^r g^{\mathsf{r}} \bmod n$ and $t = h^r$;
   (c) Finish correctly the protocol with the prover;
5. Compute the discrete logarithm of y in base g using the Pohlig-Hellman algorithm (see annex B):
$$\log_g(\mathsf{y}) = \rho \times x + \mathsf{x} \bmod \operatorname{ord}(g)$$
6. If $\log_g(\mathsf{y}) - \sum_{i=0}^{k-1} x_i 2^i \times \rho \in [0, \lfloor \frac{\lambda(n)}{2} \rfloor[$ then set $x_k = 0$;
7. Else, if $\log_g(\mathsf{y}) - \sum_{i=0}^{k-1} x_i 2^i \times \rho \in]\lfloor n/4 \rfloor, \frac{\lambda(n)}{2} + \lfloor n/4 \rfloor]$, then set $x_k = 1$;
8. Else, go to step 4;
9. Return: x_k

Fig. 2. The attacker strategy to recover x_k from $x_0, x_1, \ldots, x_{k-1}$

The total number of protocol executions to recover a ℓ-bit secret x is finally $\ell \times (1 + 1/\sqrt{n})$, since each bit requires at least one protocol execution, and the intersection of the intervals contains approximately $\sqrt{n}$ points.

The attack is no longer possible if the prover checks the correctness of n or g. However, as we will see in the next subsection, if only the quadratic residuosity is checked, a variant of the attack can be applied.

In annex B, we review some technical details related to the computation of discrete logarithms in groups of smooth order in order to provide a complete description of the attack. We also provide in section 5 practical complexity estimates for realistic parameter sizes.

3.5 Final observation: the modulus n can be prime

Let us assume that the protocol is slightly modified so that the prover checks the quadratic residuosity of g. This can be easily implemented: the verifier sends g_0 of maximal order $\lambda(n)$ and the prover sets $g = g_0^2 \bmod n$. We still assume that the prover does not make any verification on the modulus n so that it can be chosen by the cheating verifier without any restriction.

The verifier can then choose n as a **prime** number such that $n-1$ is smooth and divisible by 2^ℓ. In this case, he can still compute discrete logarithms. The generator g is a quadratic residue of maximal order $\lambda(n)/2 = (n-1)/2$. The attack we have described previously takes advantage of the short size of the mask x so it can be applied here. Indeed, by iteratively choosing the value ρ equal to $\lambda(n)/2^{i+1}$ for all the values i less than ℓ (the bit length of the secret x), the verifier is able to recover x bit by bit with approximately ℓ executions of the protocol.

In the next section, we describe an extension of the attack when the prover checks that n is not a prime number. This extension works for any unbalanced modulus, but its complexity grows exponentially with the length of the smallest factor of n.

4 Extension of the attack for an unbalanced modulus

In this section we consider the special case where the prover checks that g is chosen in the subgroup of quadratic residues of ${\mathbb{Z}_n}^*$. This can simply be done by sending g and g_0 such that $g = g_0^2 \bmod n$. We also assume that g is a quadratic residue of maximal order $\lambda(n)/2$. However we still consider that the sole check that the prover performs on n is that n is not prime. In that case, the attack of section 3.5 applies. Thus, n can be chosen by the verifier so that :

- n is unbalanced: its prime factor p is much smaller than q. With such a choice for n, the approximation of $\mathrm{ord}(g)$ by $n/4$ might not be tight, and the bias could be exploited by a dishonest verifier;
- p is small enough, and $q-1$ is smooth and divisible by a large enough power of 2, so that it is possible for the verifier to compute discrete logarithms in ${\mathbb{Z}_n}^*$.

From the value $\mathsf{y} = g_1^x g^{\mathsf{x}} = g^{\rho x + \mathsf{x}}$, the verifier can recover $X = \rho x + \mathsf{x} \bmod \mathrm{ord}(g)$, where x is uniformly distributed in $[0, \lfloor \frac{n}{4} \rfloor]$.

Let $\rho = \mathrm{ord}(g)/2$. Then X is either $\mathsf{x} \bmod \mathrm{ord}(g)$ or $\mathsf{x} + \mathrm{ord}(g)/2 \bmod \mathrm{ord}(g)$, depending on the least significant bit of x. The distribution of the X values is thus dependent on this bit. Since g is a quadratic residue of maximal order in ${\mathbb{Z}_n}^*$, we have:

$$\mathrm{ord}(g) = \frac{\lambda(n)}{2} = \frac{n}{4} - \frac{p+q-1}{4}$$

We set $\mu = (p+q-1)/4$. Thus, $n/4 = \mathrm{ord}(g) + \mu$.

The cheating verifier's strategy is detailed in figure 3. The attack consists in computing the discrete logarithm of y for each execution of the protocol, with a suitably chosen value ρ. The distribution of this value, translated according to previously computed bits, allows to infer one additional bit of the secret x.

The complexity is larger than in the previous attacks since many protocol executions are required to obtain a single bit of x. This complexity and the attack analysis are both given in annex A. With error probability $1/B$, an average of $8p\ln(B)/9$ executions of the protocol are needed for a cheating verifier to recover each bit of x from the distribution of $\log_g(\mathsf{y})$.

Figure 3 describes the attacker strategy to infer a bit of x knowing all the previous ones.

- Inputs: the bits $x_0, x_1, \ldots, x_{k-1}$ of x and a bound $\tilde{k}$ depending on the allowed error probability
- Output: the bit x_k of x

1. Generate $n = p \times q$, with p and q prime, $(p-1)/2$ and $(q-1)/2$ relatively primes, $p < 2^{20}$, $q-1$ smooth, and 2^{k+1} divides $q-1$;
2. Generate $g_0 \in {\mathbb{Z}_n}^*$ of order $\lambda(n) = (p-1)(q-1)/2$;
3. Compute $g = g_0^2 \bmod n$;
4. Set $\rho = \lambda(n)/2^{k+1}$ and compute $g_1 = g^\rho \bmod n$;
5. Set $j = 0$ and $S = 0$;
6. While $j < \tilde{k}$, do:
   (a) Execute a protocol with the prover:
      i. Send (n, g, g_1) to the prover;
      ii. Receive $\mathsf{y} = g_1^x g^{\mathsf{x}} \bmod n$, $\mathsf{t} = g_1^r g^{\mathsf{r}} \bmod n$ and $t = h^r$;
      iii. Finish correctly the protocol with the prover;
   (b) Compute the discrete logarithm of y in base g using the Pohlig-Hellman algorithm of annex B:
   $$\log_g(\mathsf{y}) = (x \times \rho + \mathsf{x}) \bmod \mathrm{ord}(g)$$
   (c) If $\log_g(\mathsf{y}) - \sum_{i=0}^{k-1} x_i 2^i \times \rho \in [0, \mu]$ then $j = j+1$;
   (d) If $\log_g(\mathsf{y}) - \sum_{i=0}^{k-1} x_i 2^i \times \rho \in [\frac{\mathrm{ord}(g)}{2}, \frac{\mathrm{ord}(g)}{2} + \mu]$, then $S = S+1$ and $j = j+1$;
7. End while;
8. If $S < \tilde{k}/2$, set $x_k = 0$,
9. Else $x_k = 1$;
10. Return: x_k

Fig. 3. The attacker strategy to recover x_k from $x_0, x_1, \ldots, x_{k-1}$ in the unbalanced case

5 Practical application of the attack

The attack has been implemented using NTL. Using RSA moduli with very small prime factors in $\lambda(n)$, a log can be computed in less than 1 second for a 2GHz PC with a 1024-bit RSA modulus. The optimum seems to be reached when using prime factors of about 5 bits.

In the cases where g is a non quadratic residue or n is prime, only one protocol run is required per secret bit, and the attack is therefore very practical: for a 160-bit secret, it requires 160 protocol runs and a few minutes of computations.

In the unbalanced case, several protocol interactions and log computations per secret bit are needed. Typically, 200 runs per bit ensures an overall success probability above 90% for a 1024-bit modulus and a 160-bit secret: only several hours of computations are required, but the secret must be extracted from the data of $160 \times 200 = 32000$ protocol runs, which might prove difficult to acquire with a real prover device.

6 Conclusion

We have described a cheating strategy for an attacker acting as a verifier in the Σ^+ proof of knowledge of discrete logarithm described in [1]. It enables to recover the full secret with essentially no computational power beyond what is required to run the protocol and after only a few iterations of it since each iteration reveals one bit of secret. We have also described variants of the attack that apply when some additional simple checks are performed by the prover, namely verifying that the modulus chosen by the verifier is indeed a composite integer and that the basis is a quadratic residue.

The correction of the Σ^+-protocol is out of the scope of this paper but it clearly appears that additional checks would probably be a sound idea. Some solutions, such as adding a proof that the RSA modulus provided by the verifier is the product of two safe primes, would drastically reduce the claimed efficiency of the protocol. Another direction would be to choose the parameter x in a large enough interval so that there is no usable bias in $\mathsf{x} \mod \operatorname{ord}(g)$, even if the parameters n and g are chosen by a dishonest verifier. While this option only adds negligible complexity to the Σ^+ protocol and thwarts all our attacks, it does not address the question of the soundness of the protocol proof.

References

1. E. Bangerter, J. Camenisch, and U. Maurer. Efficient Proofs of Knowledge of Discrete Logarithms and Representations in Groups with Hidden Order. In *PKC 2005*, LNCS 3386, pages 154–171. Springer-Verlag, 2005.
2. T. Beth. Efficient Zero-Knowledge Identification Scheme for Smart Cards. In *Eurocrypt '88*, LNCS 330, pages 77–86. Springer-Verlag, 1988.
3. J. Camenisch and M. Michels. Proving in Zero-Knowledge That a Number Is the Product of Two Safe Primes. In *Eurocrypt '99*, LNCS 1592, pages 107–122. Springer-Verlag, 1999.
4. D. Chaum, J. Evertse, and J. van de Graaf. An Improved Protocol for Demonstrating Possession of Discrete Logarithms and some Generalizations. In *Eurocrypt '87*, LNCS 304, pages 127–141. Springer-Verlag, 1988.
5. D. Chaum, J. Evertse, J. van de Graaf, and R. Peralta. Demonstrating Possession of a Discrete Logarithm without Revealing it. In *Crypto '86*, LNCS 263, pages 200–212. Springer-Verlag, 1987.

6. NESSIE consortium. *Portfolio of recommanded cryptographic primitives*, 2003. Available from `http://www.cryptonessie.org`.
7. R. Cramer. Modular Design of Secure yet Practical Cryptographic Protocol, 1997. PhD thesis, University of Amsterdam.
8. R. Cramer and I. Damgård. Zero-Knowledge Proofs for Finite Field Arithmetic or: Can Zero-Knowledge Be for Free. In *Crypto '98*, LNCS 1462, pages 424–441. Springer-Verlag, 1998.
9. I. Damgård and E. Fujisaki. A Statistically-Hiding Integer Commitment Scheme Based on Groups with Hidden Order. In *Asiacrypt 2002*, LNCS 2501, pages 125–142. Springer-Verlag, 2002.
10. W. Diffie and M. E. Hellman. New Directions in Cryptography. In *IEEE Transactions on Information Theory*, volume IT–22, no. 6, pages 644–654, november 1976.
11. E. Fujisaki and T. Okamoto. Statistical Zero Knowledge Protocols to Prove Modular Polynomial Relations. In *Crypto '97*, LNCS 1403, pages 16–30. Springer-Verlag, 1997.
12. M. Girault. Self-Certified Public Keys. In *Eurocrypt '91*, LNCS 547, pages 490–497. Springer-Verlag, 1992.
13. S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM journal of computing*, 18(1):186–208, february 1989.
14. S. C. Pohlig and M. E. Hellman. An Improved Algorithm for Computing Logarithms over GF(p) and its Cryptographic Significance. *IEEE Transactions on Information Theory*, IT–24(1):106–110, january 1978.
15. J. M. Pollard. Monte Carlo Methods for Index Computation (mod p). *Mathematics of Computation*, 32(143):918–924, July 1978.
16. G. Poupard and J. Stern. Security Analysis of a Practical "on the fly" Authentication and Signature Generation. In *Eurocrypt '98*, LNCS 1403, pages 422–436. Springer-Verlag, 1998.
17. C. P. Schnorr. Efficient Identification and Signatures for Smart Cards. In *Crypto '89*, LNCS 435, pages 235–251. Springer-Verlag, 1990.
18. P. C. van Oorschot and M. J. Wiener. On Diffie-Hellman Key Agreement with Short Exponents. In *Eurocrypt '96*, LNCS 1070, pages 332–343. Springer-Verlag, 1996.

A Analysis of the unbalanced modulus case

In the following we show that the attack, described in section 4 in the case of prime modulus, can also applied if the modulus is unbalanced. In that case, we will show that its complexity grows exponentially with the length of the smallest factor.

We recall that the modulus n is unbalanced and that g is a quadratic residue of maximal order. In the following, we analyze the attack in detail. We briefly recall some notations already given in section 4. Let x_0 denote the least significant bit of x, *i.e.* $x_0 = x \bmod 2$, X^0 the value of X for $x_0 = 0$ and X^1 the value of X for $x_0 = 1$. Since g is a quadratic residue of maximal order in $\mathbb{Z}_n{}^*$, we have:

$$\mathrm{ord}(g) = \frac{\lambda(n)}{2} = \frac{n}{4} - \frac{p+q-1}{4}$$

We set $\mu = (p + q - 1)/4$. Thus, $n/4 = \mathrm{ord}(g) + \mu$.

For $x_0 = 0$, $X^0 = \mathsf{x}$ is uniformly distributed in the interval $[0, \lfloor \frac{n}{4} \rfloor] = [0, \mathrm{ord}(g) + \lfloor \mu \rfloor]$. Taking the values modulo $\mathrm{ord}(g)$, we have :

$$\begin{aligned}\Pr(X^0 \in [0, \lfloor \mu \rfloor]) &= \Pr(\mathsf{x} \in [0, \lfloor \mu \rfloor] \cup [\mathrm{ord}(g), \mathrm{ord}(g) + \lfloor \mu \rfloor]) \\ &= \frac{2\lfloor \mu \rfloor}{\mathrm{ord}(g) + \lfloor \mu \rfloor} \approx \frac{8\mu}{n}\end{aligned}$$

$$\Pr(X^0 \in [\lceil \mu \rceil, \mathrm{ord}(g)]) \approx 1 - \frac{8\mu}{n}$$

We can easily infer the distribution for $x_0 = 1$ with a circular shift of width $\mathrm{ord}(g)/2$. Since $X^1 = \mathsf{x} + \mathrm{ord}(g)/2$ is uniformly distributed in the interval $[\frac{\mathrm{ord}(g)}{2}, \frac{3\,\mathrm{ord}(g)}{2} + \lfloor \mu \rfloor]$, we thus obtain

$$\begin{aligned}\Pr(X^1 \in [0, \mathrm{ord}(g)/2] \cup [\mathrm{ord}(g)/2 + \lceil \mu \rceil, \mathrm{ord}(g)]) &= \Pr(\mathsf{x} \in [\mathrm{ord}(g)/2 + \lceil \mu \rceil, 3\,\mathrm{ord}(g)/2]) \\ &= \frac{\mathrm{ord}(g) - \lceil \mu \rceil}{\mathrm{ord}(g) + \lfloor \mu \rfloor} \approx 1 - \frac{8\mu}{n}\end{aligned}$$

$$\Pr(X^1 \in [\mathrm{ord}(g)/2, \mathrm{ord}(g)/2 + \lfloor \mu \rfloor]) \approx \frac{8\mu}{n}$$

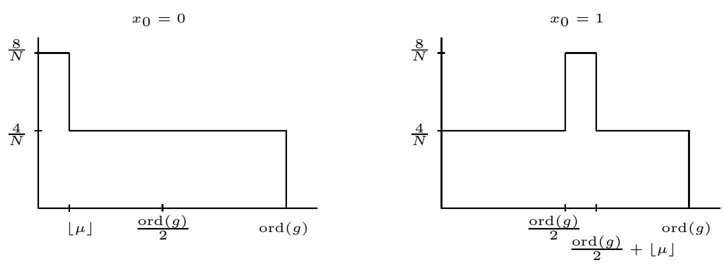

Fig. 4. The distribution of X^0 and X^1.

The verifier should run the protocol several times to distinguish these two distributions. Each time the value X obtained is not in the intervals $[0, \lfloor \mu \rfloor]$ or $[\mathrm{ord}(g)/2, \mathrm{ord}(g)/2 + \lfloor \mu \rfloor]$ the verifier gains no information. Accordingly we consider only the values X in these intervals and try to distinguish $x_0 = 0$ from $x_0 = 1$. We set $\tilde{X} = 0$ if $X \in [0, \lfloor \mu \rfloor]$ and $\tilde{X} = 1$ if $X \in [\mathrm{ord}(g)/2, \mathrm{ord}(g)/2 + \lfloor \mu \rfloor]$. We ignore the other cases so that we keep in average only 3μ values amongst $\mathrm{ord}(g) + \mu$. Depending on the bit x_0, $\tilde{X}$ has the following distribution:

$$\begin{aligned}&\text{if } x_0 = 0,\ \Pr(\tilde{x} = 0) = \frac{2}{3} \quad \text{and} \quad \Pr(\tilde{x} = 1) = \frac{1}{3} \\ &\text{if } x_0 = 1,\ \Pr(\tilde{x} = 0) = \frac{1}{3} \quad \text{and} \quad \Pr(\tilde{x} = 1) = \frac{2}{3}\end{aligned}$$

Let $\tilde{k}$ be the number of values collected by the verifier lying in the suitable ranges. Let $S^b_{\tilde{k}}$ the sum of the $\tilde{x}$ depending on the value b of the bit x_0. The Chernoff bound shows that, for every $\varepsilon > 0$,

$$\Pr\left(\frac{S^0_{\tilde{k}}}{\tilde{k}} - \frac{1}{3} \geq \varepsilon\right) \leq e^{-\tilde{k}\varepsilon^2 \times \frac{3}{4}}$$

and

$$\Pr\left(\frac{S^1_{\tilde{k}}}{\tilde{k}} - \frac{2}{3} \leq \varepsilon\right) \leq e^{-\tilde{k}\varepsilon^2 \times \frac{1}{6}}$$

If ε is the sample mean of the two distributions, *i.e.* $\varepsilon = 1/2$, this allows us to have a bound on the number of values needed so that the error value is not too large. For an error probability less than $1/B$, then the number $\tilde{k}$ of collected $\tilde{x}$ values should be such that $\tilde{k} \geq 16\ln(B)/3$.

Taking into account the number of unused values X, we obtain that the total number of verifications to learn 1 bit of information with probability $1/B$ is:

$$\begin{aligned} k &\geq 16\frac{\ln(B)}{3} \times \frac{\mathrm{ord}(g) + \mu}{3\mu} \\ &\geq \frac{16\ln(B)}{3} \times \frac{n}{3(p+q-1)} \\ &\geq \frac{16p\ln(B)}{9(1+\frac{p-1}{q})} \\ &> \frac{8p\ln(B)}{9} \end{aligned}$$

Practical results. Such a bound on the number of runs needed to learn one bit of information allows us to estimate the complexity of the attack depending on p and q. If p is really small, for example if $p = 3$, we obtain $k \geq 26$ for an error probability per bit equal to $1/1000$.

When p is larger, the number of runs explodes. Indeed, the number of queries strongly depends on the length of p and becomes too large as soon as p is larger than say 2^{30}. For such a value, and for a 256 bits secret x, the total complexity of the attack can be approximated by 2^{40}, for an error probability for each bit of x which is $1/B = 1/1000$.

Using Additional Information in X. To improve the overall success probability of the attack, we can analyze what happens when a bit was guessed incorrectly. In that case, when treating the next bit, one gets the distributions of figure 4, with a circular shift of $\mathrm{ord}(g)/4$. Irrespectively of the correctness of the previous guess, the two candidate distributions for X are equal up to a shift by $\mathrm{ord}(g)/2$. As a consequence, the distribution of $2X \mod \mathrm{ord}(g)$ can take two values: a distribution D_1 when the previous bit was guessed correctly, and a distribution D_2 otherwise. D_1 and D_2 have the same shape as the distributions of figure 4, with $\mu = 2\frac{p+q-1}{4} = \frac{p+q-1}{2}$. Because of the multiplication by 2, the peak is only 3/2 as high as the rest of the distribution.

These remarks can be used to add new experiments regarding bit x_i when performing the experiments on bit x_{i+1}. D_1 and D_2 are harder to distinguish than the distributions of X^0 and X^1; therefore, the new experiences are less conclusive, and "weight" less than the first series; the weight ratio is $\ln(3/2)/\ln(2)$. This is partly compensated by the higher probability to land in the peaks of distributions D_1 and D_2, which are twice as wide as for the distributions of X^0 or X^1. Overall, with the same success probability per bit, these additional experiences save up to 54% of the log computations, depending on μ. The most attractive case is when the two μ-wide peaks of distributions D_1 and D_2 do not overlap, in which case the saving ratio is $\frac{2\ln(3/2)}{2\ln(3/2)+\ln(2)} \approx 0.54$.

The algorithm finally obtained is described figure 5.

B Practical computation of discrete logarithms in groups of smooth order

In the following we show how to compute discrete logarithms when the order's factorization of the group element is unknown, but only small factors are known.

Let $\mathcal{G}$ be a multiplicative group. We do not assume any specific property of this group in this section. Let g be an element of multiplicative order ω.

Generic algorithms to compute discrete logarithms, such as Baby step-Giant step or Pollard rho and lambda methods [15, 18] have complexity $O(\sqrt{\omega})$. However, in some cases, more efficient techniques apply. The well-known Pohlig-Hellman algorithm [14] takes advantage of the factorization of the order ω when it is applicable. If we choose the group parameters such that this order is smooth, this algorithm enables to compute discrete logarithms efficiently.

We now describe a variant strongly inspired from the original Pohlig-Hellman algorithm. We note

$$\omega = \prod_{i=1}^{k} p_i^{e_i} \quad \text{with} \quad \begin{cases} \forall i \in [1,k] \;\; p_i \;\text{ is a prime integer} \\ \forall i \in [1,k] \;\; e_i \in \mathbb{N}^* \\ 1 \leq i < j \leq k \Rightarrow p_i < p_j \end{cases}$$

and we consider the algorithm of figure 6.

Note that if we use this algorithm with $\ell = k$, it just computes discrete logarithms using the Pohlig-Hellman idea, performing the Chinese remainder computation whenever it is possible. We can also use it with $\ell < k$; in this case we can compute some partial information about the discrete logarithms. This may have important consequences when some optimizations such as so-called short exponents, *i.e.* exponents much smaller than the order ω but larger than 160 bits are used for efficiency reasons. In such a situation, the complete factorization of the order of g may be unknown but enough small factors p_i may still enable to recover some secrets.

- Inputs: a bound $\tilde{k}$ depending on the allowed error probability
- Output: the secret x

1. Generate $n = p \times q$, with p and q prime, $p = 3 \mod 4$, p small, $p-1$ and $q-1$ smooth, $(p-1, q-1) = 2$ and 2^{k+1} divides $q-1$;
2. Generate $g_0 \in \mathbb{Z}_n{}^*$ of order $\lambda(n) = (p-1)(q-1)/2$;
3. Compute $g = g_0^2 \bmod n$;
4. $S[i] = 0$, $i = 0, \ldots, k-1$
5. $X[j] = 0$, $j = 0, \ldots, \tilde{k}$
6. $z = 0$, $\eta = \mathrm{ord}(g)/2$
7. For $i = 0, \ldots, k-1$, do
   (a) Set $\rho = \lambda(n)/2^{i+2} = \mathrm{ord}(g)/2^{i+1}$ and $g_1 = g^\rho \bmod n$;
   (b) While $j < \lfloor \tilde{k} \times \mathrm{ord}(g)/(2\mu) \rfloor$, do:
      i. Execute a protocol run and extract
      $$\log_g(\mathsf{y}) = (x \times \rho + \mathsf{x}) \bmod \mathrm{ord}(g)$$
      ii. $X[j] = \log_g(\mathsf{y}) - z \times \rho$
      iii. If $i > 0$, do
         A. If $2X[j] \in [\max(0, 2\mu - \eta), \min(2\mu, \eta)]$ then $S[i-1] -= \ln(3/2)$;
         B. If $2X[j] \in [\max(\eta, 2\mu), \min(\mathrm{ord}(g), \eta + 2\mu))]$, then $S[i-1] += \ln(3/2)$;
   (c) End while;
   (d) If $i > 0$, do
      i. If $S[i-1] < 0$, set $x_{i-1} = 0$;
      ii. Else $x_{i-1} = 1$;
      iii. $z = z + x_{i-1} 2^{i-1}$
   (e) While $j < \lfloor \tilde{k} \times \mathrm{ord}(g)/(2\mu) \rfloor$, do:
      i. If $X[j] \in [0, \mu]$ then $S[i] -= \ln(2)$;
      ii. If $X[j] \in [\eta, \eta + \mu]$, then $S[i] += \ln(2)$;
   (f) End while;
8. End For;
9. If $S[k-1] < 0$, set $x_{k-1} = 0$;
10. Else $x_{k-1} = 1$;
11. Return: $x = \sum_{i=0}^{k-1} x_i 2^i$

Fig. 5. The attacker improved strategy to recover x in the unbalanced case

Theorem 1. *On input $y \in \langle g \rangle$ and $\ell \in [1, k]$, the algorithm of figure 6 computes $X = \log_g(y) \bmod \prod_{i=1}^{\ell} p_i^{e_i}$. The time complexity is $O\left(\sum_{i=1}^{\ell} e_i \times \sqrt{p_i}\right)$.*

Proof. The justification of the result is done recursively. For any value of indexes i and j, we have, just before line "i." the following relations:

- $P = \prod_{\alpha=1}^{i-1} p_\alpha^{e_\alpha} \times p_i^{j-1}$
- $G = g^P$
- $\Omega = \omega/P$

1. <u>input</u>: $y \in \langle g \rangle$
2. <u>initialization</u>: $Y = y$, $G = g$, $\Omega = \omega$, $P = 1$, $X = 0$
3. for i from 1 to ℓ do
   (a) for j from 1 to e_i do
      i. $\Omega = \Omega / p_i$
      ii. $z = \log_{G^\Omega}\left(Y^\Omega\right)$
      iii. $Y = Y/G^z$
      iv. $G = G^{p_i}$
      v. $X = X + P \times z$
      vi. $P = P \times p_i$
4. <u>return</u>: X

Fig. 6. A variant of the Pohlig-Hellman algorithm to compute discrete logarithms

- $X = x \bmod P$
- $Y = g^{(x \text{ div } P) \times P} = y/g^X$

After execution of line "i.", the new value of Ω is $\Omega = \omega/(P \times p_i)$. Then, in line "ii.", we have

$$G^\Omega = g^{P \times \frac{\omega}{P \times p_i}} = g^{\omega/p_i}$$

and
$$Y^\Omega = g^{(x \text{ div } P) \times P \times \frac{\omega}{P \times p_i}} = g^{(x \text{ div } P) \times \frac{\omega}{p_i}}$$

so the computation of $z = \log_{G^\Omega}\left(Y^\Omega\right)$ leads to

$$z = \log_{g^{\omega/p_i}}\left(\left(g^{\omega/p_i}\right)^{x \text{ div } P}\right) = (x \text{ div } P) \bmod p_i$$

An important fact for the complexity of the algorithm is that z is an integer in the range $[0, p_i - 1]$ because g^{ω/p_i} has multiplicative order p_i. Consequently, if p_i is small, we can use generic discrete logarithm algorithms with running time $O(\sqrt{p_i})$ to efficiently compute z.

Then, after computation "iii.", we have

$$Y = g^{(x \text{ div } P) \times P - (x \text{ div } P) \bmod p_i} = g^{(x \text{ div } (P \times p_i)) \times (P \times p_i)}$$

After the next computation, $G = \left(g^P\right)^{p_i} = g^{P \times p_i}$ and then

$$X = x \bmod P + P \times z = x \bmod P + P \times ((x \text{ div } P) \bmod p_i) = x \bmod P \times p_i$$

and finally
$$P = \prod_{\alpha=1}^{i-1} p_\alpha^{e_\alpha} \times p_i^{j-1} \times p_i = \prod_{\alpha=1}^{i-1} p_\alpha^{e_\alpha} \times p_i^j$$

The result X which is returned is $X = x \bmod \prod_{\alpha=1}^{i} p_\alpha^{e_\alpha}$.

The main computation is the evaluation of z on line "ii.". Its complexity is $O(\sqrt{p_i})$ so the global time complexity of the algorithm is $O\left(\sum_{i=1}^{\ell} e_i \times \sqrt{p_i}\right)$.

□