Diffie-Hellman is as Strong as Discrete Log for Certain Primes

Bert den Boer *

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

Abstract

Diffie and Hellman proposed a key exchange scheme in 1976, which got their name in the literature afterwards. In the same epoch-making paper, they conjectured that breaking their scheme would be as hard as taking discrete logarithms. This problem has remained open for the multiplicative group modulo a prime P that they originally proposed. Here it is proven that both problems are (probabilisticly) polynomial-time equivalent if the totient of P-1 has only small prime factors with respect to a (fixed) polynomial in ${}^2logP$.

There is no algorithm known that solves the discrete log problem in probabilistic polynomial time for the this case, i.e., where the totient of P-1 is smooth. Consequently, either there exists a (probabilistic) polynomial algorithm to solve the discrete log problem when the totient of P-1 is smooth or there exist primes (satisfying this condition) for which Diffie-Hellman key exchange is secure.

Introduction

Let P be a prime and g be a generator of the multiplicative group $GF(P)^*$. Let f be an element of $GF(P)^*$. Solving for x

$$(1) \qquad f = g^x \bmod P,$$

*Supported in part by the Netherlands Organization for Scientific Research (NWO).

S. Goldwasser (Ed.): Advances in Cryptology - CRYPTO '88, LNCS 403, pp. 530-539, 1990.

is called the *Discrete Logarithm Problem* (D.L. problem). Until now, no general purpose algorithm has appeared that computes x mod $P\text{-}1$ in probabilistic polynomial time (see [Odl]). In [Poh], an algorithm is given which solves the D.L. problem in polynomial time for the case where all prime factors of $P\text{-}1$ are smaller than some constant B. The time remains polynomial if we take $B = q({}^2logP)$, for some fixed polynomial $q(.)$. Numbers with this property will be called *smooth*(with respect to P and q).

Based on the difficulty of solving the general D.L. problem a key exchange protocol is proposed in [DH]: The first step is to establish a prime P and a generator g of $GF(P)^*$ common for all participants. Each party j chooses randomly a secret number z_j and computes $f_j = g^{z_j}$. The f_j are made public. The public numbers are sufficient to establish a secret and common key for each pair of participants. For example let us assume that the first party has secret x and computed

(2) $$f_1 = g^x \bmod P.$$

and similarly the second party has secret y and computed

(3) $$f_2 = g^y \bmod P.$$

The first party computes $f_2{}^x$ and the second party computes $f_1{}^y$. Thus both parties obtain

(4) $$f_3 = g^{xy} \bmod P,$$

which they can use as a common secret key. For a passive eavesdropper to determine this key, he must find f_3 satisfying (4) with suitable x and y satisfying (2) and (3) where the gathered data is just

$$(P,g,f_1,f_2).$$

This problem will be called the *D.H. problem.* An obvious way to solve it is to solve a D.L. problem with input (P,g,f_1) and then to use the output x

to compute f_3 by $f_3 = f_2^x$. The critical question is can f_3 be found without obtaining (much) information about x or y in the process. In [DH] the authors conjecture that solving x or y is basicly the only way to solve the D.H. problem. More precisely they conjecture that the D.H. problem is hard if the D.L. problem is hard. A sufficient way to prove this is showing there exists an algorithm to solve the D.L. problem comprising of a (probabilistic) polynomial number of calls to an algorithm which solves the D.H. problem (here after called a *D.H. oracle*) and a (probabilistic) polynomial number of "elementary" operations.

In this paper we present such an algorithm for the case where the totient of P-1 is smooth with respect to P and $q(.)$. In the literature no probabilistic time algorithm $\mathfrak{R}$ exists to solve the D.L. problem for primes P for which the totient of P-1 is smooth (notice that P-1 itself has to be smooth for Pohlig-Hellman algorithm to be polynomial-time, see[Poh]). From our design follows that either such an algorithm $\mathfrak{R}$ exists or primes P exist for which the D.H. problem is hard. In the latter case primes P for which the biggest prime factor of P-1 is bigger than P^e for some fixed positive real number e serve as candidates (while the double totient of P is smooth).

Main idea

Let B be a number which depends polynomially on the logarithm of P. It always holds that P-1 has a unique factorization MN, where the prime factors of M are smaller than B and the prime factors of N are bigger (or equal) than B. Assume also that the totient of P-1 is smooth. Given a generator g of the multiplicative group modulo P and a member f of that group we are interested in solving x from (1). It suffices to compute x modulo M and x modulo N and use them in a Chinese Remainder Algorithm. A solution z of the equation

$$f^N = g^{zN} \bmod P$$

is equal to x mod M. This number can be computed in polynomial time with techniques similar to Pohlig-Hellman[Poh].

Replacing N by M in the last equation and solving the new equation would give us x modulo N . At this moment it is not (publicly) known how to solve this in polynomial time. What we do is exploiting the D.H.oracle to solve that new equation. Because the totient of $P\text{-}1$ is smooth, each prime factor Q of N appears with multiplicity 1 (otherwise Q would be a prime factor of the totient of $P\text{-}1$) and moreover it holds that $Q\text{-}1$ is smooth. We proceed computing x modulo N by computing x modulo Q for each prime divisor of N.

This algorithm requires a generator h of the multiplicative group $GF(Q)^*$, which can be found in probabilistic polynomial time, [Rie] (in a practical setting h is already constructed while constructing the big prime P). Either $x = 0$ mod Q or $x = h^y$ mod Q for some y. The first possibility appears iff $f^L = 1$ mod P, where $LQ = P\text{-}1 = MN$. In this (unlikely) case we are lucky, having found the answer (x mod Q) already. In the other case we proceed by computing the unknown y mod $Q\text{-}1$.

To compute y it is sufficient to compute y modulo any prime power divisor of $Q\text{-}1$ because we can combine these answers with the Chinese Remainder Algorithm to get y modulo $Q\text{-}1$. We will only show how to compute y modulo a prime divisor p of $Q\text{-}1$.

Let l be $(Q\text{-}1)/p$ and compute $U = g^{x^l}$. This can be done by less than $2\ {}^2log(l)$ calls to the D.H.oracle. This indeed can be done because on input (g, g^{x^a}, g^{x^b}) the D.H.oracle should output $g^{x^{a+b}}$. Next one computes U^L and compares this with $g^{Lh^{la}}$ for $0 \leq a < p$. There is bound to be equality for one of the choices of a and for this a it holds that $a = y$ modulo p .

The above mentioned equality occurs because it is equivalent to the equality $Lh^{yl} = Lh^{al}$ mod $P\text{-}1$ and this is equivalent to the equality $h^{yl} = h^{al}$ mod Q. This last equality is equivalent to $yl = al$ mod $Q\text{-}1$ and finally this is equivalent to $y = a$ mod p.

Combining these answers for all prime divisors of $Q\text{-}1$ with the Chinese Remainder Algorithm we get a solution for y mod $Q\text{-}1$ and we compute $x = h^y \bmod Q$. Repeating this for each prime divisor Q of N and using the the Chinese Remainder Algorithm again we establish x mod N. Combining this with the already established value for x mod M we have arrived at the solution of equation (1). In the actual algorithm in the next section we do not mind about multiplicities of prime factors and we will also exploit the birthday attack. Furthermore we solve y modulo Q-1 without the Chinese Remainder Algorithm.

The main algorithm

In what follows we describe our algorithm to solve the D.L. problem ,an algorithm which is allowed to make a polynomial number of calls to a D.H. oracle and a polynomial number of elementary steps. At the end of this section we will be specific about these numbers.

Let $q(.)$ be a fixed polynomial. Let P be prime and $P\text{-}1 = MN$, where M, resp. N has small, resp. big prime factors with respect to $B = q(log(P))$ (like in the previous section). We want to solve equation (1). We already remarked that solving x modulo M can be done in polynomial time. In practical settings we can assume that the factorization of N is given (otherwise finding a generator modulo P would be difficult). If the factorization of N is not given we can find the factors of N in probabilistic polynomial time using Pollard-p-1 method [Pol] in an adapted form (details left to the reader). As we remarked in the previous section N is squarefree.

Let Q be prime divisor of N. We will establish w defined by x modulo Q. Let us define $L =(P\text{-}1)/Q$. The algorithm to compute w requires a generator h of the multiplicative group $GF(Q)^*$, which can be found in probabilistic polynomial time, [Rie] (in a practical setting h is already constructed while constructing the big prime P). Either $w = 0$ mod Q or $w = h^y$ mod Q. The first possibility appears iff $f^L = 1$ mod P. In this (unlikely) case we are done. In the other case we proceed by computing the unknown y mod Q-1.

Now we will describe the residues which has to be computed in our algorithm. Given is the factorization $Q\text{-}1 = \prod_{j=1}^{k} p_j$, where $p_i \le p_{i+1}$ and $p_k \le B$. Define for $j = (-1),(0),1,\ldots,k\text{-}1,(k)$

$$l_0 = 1,$$

$$l_j = \prod_{i=1}^{j} p_i \quad ,$$

$$m_j = (Q\text{-}1)/p_{j+1} \; ,$$

$$D_j = \prod_{i=j+2}^{k} p_i \; ,$$

$$D_{k\text{-}1} = 1 \; ,$$

$$r_j = \lceil \sqrt{p_j} \rceil,$$

$$U_j = g^{x^{l_j}} \text{ modulo } P,$$

$$t_{bj} = (p_{j+1} - br_{j+1})m_j \text{ and}$$
$$T_{bj} = g^{Lh^{t_{bj}}} \text{modulo } P$$
$$\text{for } 0 \le b \le \lfloor (p_j/r_j) \rfloor,$$

$$S_{aj} = h^{am_j} \text{ modulo } Q \text{ for } 0 \le a < r_j,$$

$$y_{k\text{-}1} = 0 \; ,$$

$$C_j = h^{y_j l_j} \text{ modulo } Q.$$

The algorithm starts with computing U_{j+1} from U_j using the D.H.oracle. All intermediate U_j have to be stored. The residues T_{bj} need to be stored for each different prime divisor p_j of $Q\text{-}1$. The residues S_{aj} may need to be stored (especially if p_j^2 is a divisor of $Q\text{-}1$). We compute z_j starting with $j = k\text{-}1$ down to 0 and y_j starting from $j = k\text{-}2$ down to $\text{-}1$ as follows. For $j = k\text{-}1$ down to 0 we search for a and b for which the equation

$$(5) \qquad (U_j^{C_j L})^{S_{aj}} = T_{bj}$$

holds. Define z_j by $a+br_{j+1}$ and $y_{j\text{-}1}$ by $y_j+z_jD_j$. Equation (5) always has a solution and finally we can compute w because it is equal to $h^{Q\text{-}1\text{-}y_{\text{-}1}}$ modulo Q.

Now we will briefly sketch why this algorithm computes x mod Q. After we checked that Q does not divide x we may assume that $x = h^y + Qr \bmod MN$ for some y and r. By induction we will show that $y + y_i = 0 \bmod D_i$ for $i = k\text{-}1,\dots,\text{-}1$.

This is trivially true for $i = k\text{-}1$. Consider the left side of (5). This is equal

$$g^{(h^y + Qr)^{l_j} h^{y_j} L h^{am_j}} .$$

The exponent can be written as

$$(h^{yl_j} + Qt)\, h^{y_j l_j} L h^{am_j}$$

for some t. The right side of (5) is equal to

$$g^{L h^{(p_{j+1} - b r_{j+1})}} .$$

Equality (5) holds iff the exponents are equal mod MN. This holds iff

$$h^{y\, l_j + y_j\, l_j + a\, m_j} = h^{(p_{j+1} - b\, r_{j+1}) m_j} \bmod Q.$$

This last equality holds iff

$$(6) \qquad l_j(y + y_j) + a\, m_j = (p_{j+1} - b\, r_{j+1})\, m_j \bmod Q\text{-}1.$$

Under the induction hypothesis $y + y_j = c\, D_j$ for some c. Because $l_j\, D_j$ equals m_j equation (6) holds iff $c + a = p_{j+1} - b\, r_{j+1}\ mod\, p_{j+1}$. Because of the chosen range for (a,b) such an equality will occur. For this pair (a,b) we define $z_j = a + b_j$ and $y_{j\text{-}1} = y_j + z_j\, D_j$. Now it holds that

$$\begin{aligned} y + y_{j\text{-}1} &= y + y_j + (a + b\, r_{j+1})\, D_j = \\ &= (c + a + b\, r_{j+1})\, D_j = \\ &= 0 \bmod p_{j+1}\, D_j. \end{aligned}$$

Because $p_{j+1}\, D_j = D_{j\text{-}1}$, this proves our induction step and ends the sketch of the proof.

We repeat this algorithm for each (big) divisor Q of N and use the Chinese Remainder Algorithm to find x modulo N. After that another combination with x modulo M establishes the solution x (modulo $P\text{-}1$) of our original problem (equation (1).)

In the following theorem we assume that the factorization of $P\text{-}1$ is given and also that generators of the multiplicative groups $GF(Q)^*$ are given for each big prime factor Q of $P\text{-}1$. Furthermore we require that the D.H. oracle gives the right answer for each question.

Theorem : An algorithm to solve the D.L. problem for a prime P for which the totient of $P\text{-}1$ is smooth with respect to P and q requires order $log^2(P)$ calls to the D.H.oracle and order $log^2(P)\sqrt{q(log(P))}$ multiplications. This algorithm requires order $log^2(P)\sqrt{q(log(P))}$ of bits of memory space.

The first two assumptions of the three assumptions in the paragraph just preceding our theorem can be dropped and we still have (probabilistic) polynomial-time equivalence. This is also the case if we weaken the last of the three assumptions (because of the algebraic structure we can construct "majority answers").

Conclusion

At this moment no efficient algorithm to compute the Discrete Logarithm is known for the case where the totient of P contains a prime factor bigger than P^e, where e is some fixed positive real number. So we could safely use a Diffie-Hellman key exchange for the subcase where the biggest prime factor of the double-totient is smaller than some fixed polynomial in the logarithm of P

An interesting property of our algorithm to solve the D.L. problem using a D.H. oracle is that it mimics the algorithm to solve the D.L. problem in polynomial time without a D.H. oracle for the case where $P-1$ itself is smooth. It is conceivable that a polynomial-time algorithm to solve the D.L. problem for the case where the totient of $P-1$ (the *double totient case*) is smooth may enable us to design an algorithm to solve the D.L. problem in the triple totient case in polynomial time and a polynomial number of calls to a D.H. problem. If this goes on for higher and higher totients we at last have proved that either the general D.L. problem has a polynomial-time solution or there exists primes for which the D.H. problem is hard.

References

[DH] Diffie, W. and M.E. Hellman, New directions in cryptography, IEEE Trans. Inf. Theory, IT-22, pp. 644-654, Nov. 1976.

[Odl] Odlyzko, A.M., Discrete logarithms in finite fields and their cryptographic significance, Advances in Cryptology: Proc. Eurocrypt '84, Lecture Notes in Computer Science 209, Springer, Berlin etc., pp. 224-314, 1985

[Poh] Pohlig, S.C. and M.E. Hellman, An improved algorithm for computing logarithms over GF(p) and its cryptographic significance, IEEE Trans. Inf. Theory, IT-24, pp.106-110, Jan 1978.

[Pol] Pollard, J.M., Theorems on Factorization and Primality testing, Proc. Cambr. Philos. Soc., 76, pp 521-528, 1974

[Rie] Riesel, H., Primality Testing and Factorisation, Birkhauser, Boston, 1985