Diffie-Hellman and the Discrete Logarithm

Z. Bilkin, K. Levy, C. Seaman
(all work is our own)

Introduction

Our Groups problem was to attack a Diffie-Hellman style key exchange, a key exchange which establishes a key between two parties in the following manner:

A sufficiently large prime number p is generated. It is clear that the integers residues modulo p form a the finite field (called $\mathbb{Z}_p)$ with the normal modular addition and subtraction. Now if one examines the $p-1$ non-zero elements, $\mathbb{Z}_p^\times$, it is clear that they form a group. The fact that this group is cyclic is a little less clear.

Given $\mathbb{Z}_p^\times$ the Finite Structure Theorem for Abelian Groups tells us that where $d_1|d_2|\ldots|dn$ and $p-1 = d_1 d_2 \ldots dn.$ Now clearly it follows that if $x \in \mathbb{Z}_p^\times$ it is a solution to $x^{d_n} - 1 = 0$ which in turn has at most d_n solutions. Thus $d_n = p-1$

Thus an element α can be found such that α generates $\mathbb{Z}_p^\times$. Now the Diffie-Hellman Protocol functions by making some such prime p and α public. Each party picks some "random" number from 1 to the order of α (which is $p-1$ if α is a generator of $\mathbb{Z}_p^\times$) call these numbers a and b. They keep these numbers secret but then calculate and broadcast α^a and α^b to their respective counterpart. Now each counterpart has either $a \text{ and } \alpha^b$ or $a \text{ and } \alpha^a$ so they compute $(\alpha^a)^b \text{ and } (\alpha^b)^a$ which are clearly equal.

Now the security of this rests on the unproven assumption that solving the discrete log is hard. That is if an eavesdropper intercepts α^b and α^a and knows the public p and α there is no obvious way of computing a and b.(or α^{ab} for that matter.)

The specific problem we set out to crack used the following 62-digit modulus ($62 * \log_2 10 \doteq 206$ bit) and similarly big powers of the group generator.

$p = 32170146391986017710904672999863498684363935740291724566674199$
$\beta_1 \equiv 5^a (mod p) = 24424105714444366547244972571550840662055244077162355600491$
$\beta_2 \equiv 5^b (mod p) = 79417598523393217148837918455130125749445849993750234415500 4$

(since $5^{\frac{p-1}{2}} = 1(mod p)$ we see that 5 does not generate the multiplicative group. Since $\frac{p-1}{2}$ is itself prime, $\frac{p-1}{2} = ord(5)$. This "complication" is discussed in the section on the Index Calculus.)

Attacks on the Discrete Log Problem

Overview

There are several attack methods for discrete log problems. Each of them has some advantages under special circumstances. The followings are some of them which we didn’t use.

- Brute Force
- Pohlig-Hellman
- Baby-Step Giant-Step
- Lambda (Kangaroo)

Briefly, we will explain why we didn’t use these methods.

Brute Force

In this method every possible key is being tried until the correct one is found. To calculate required time to solve our problem by this method, we’ve done a small measurement of speed.

The laptop used for speed-test have following specifications:
Processor: Intel Centrino 1.86GHz
Memory: 1.5 GB RAM

```
#this program is written for testing the number of calculations per second
p=32170146391986017710904672999863498684363935740291724566674199
5^a=244241057144443665472449725715508406620552440771362355600491
5^b=794175985233932171488379184551301257494458499937502344155004
x=0
m=1
while x<(10**8):
    x=x+1
    m=(5*m)%p
    if (m==(5^a or 5^b)):
        print "5^x=",m," for x(a or b)=",x
```

Calculation of 10^8 times modular exponentiation takes 3 minutes with the algorithm shown above.
By using this result we can estimate the time required for solving our problem.

3 minutes for 10^8 calculations
30 minutes for 10^9 calculations
5 hours for 10^10   calculations
400 days(about 1 years) 2*10^13 calculations

For our problem we need 10^62/(2*10^13)=5*10^48 years to solve it by this method. It is clear that why we didn’t use this method.

Pohlig-Hellman

This method requires factorization of p-1 if the base of logarithm is primitive root. Otherwise it requires factorization of the order of the group.

Assume our discrete log problem is $\beta=\alpha^x \pmod p$, and we have the following;

$$p-1=\prod_i q^{r_i} \quad , \text{where } q_i \text{ ‘s are factors of } p-1$$

Let q^{r_i} be one of the factors. We can calculate $L_\alpha(\beta) \pmod{q^{r_i}}$. If this can be done, the answers can be combined using the Chinese remainder theorem to find the original discrete logarithm.

In our problem p-1 is a 62-digits number. We tried to factor it by first one million primes. However p-1 has only 2 as a factor within first one million primes. Then we checked (p-1)/2 whether it is prime or not. It is turn out that (p-1)/2 is prime. This means that we can not use Pohlig-Hellman method since p-1 have only 2 and (p-1)/2, which is a big prime, as factors.

Baby-Step Giant-Step

This method requires $O(\sqrt{p})$ running time and $O(\sqrt{p})$ space. This deterministic algorithm requires the construction of two arrays of group elements.

Let $\beta=\alpha^x \pmod p$ is a discrete log problem.
In this method first of all it is chosen an integer N such that $N2 \geq p-1$.
Then these two lists are formed:

$\alpha^j \pmod p$ for $0 \leq j< N$
$\beta\, \alpha^{-Nk} \pmod p$ for $0\leq k< N$

Then it is looked for a match between the two lists. If one is found, then

$$\alpha^j\equiv\beta\, \alpha^{-Nk}$$

so $\alpha^{j+Nk} \equiv \beta$. Therefore, $x=j+Nk$ solves the discrete log problem (1)

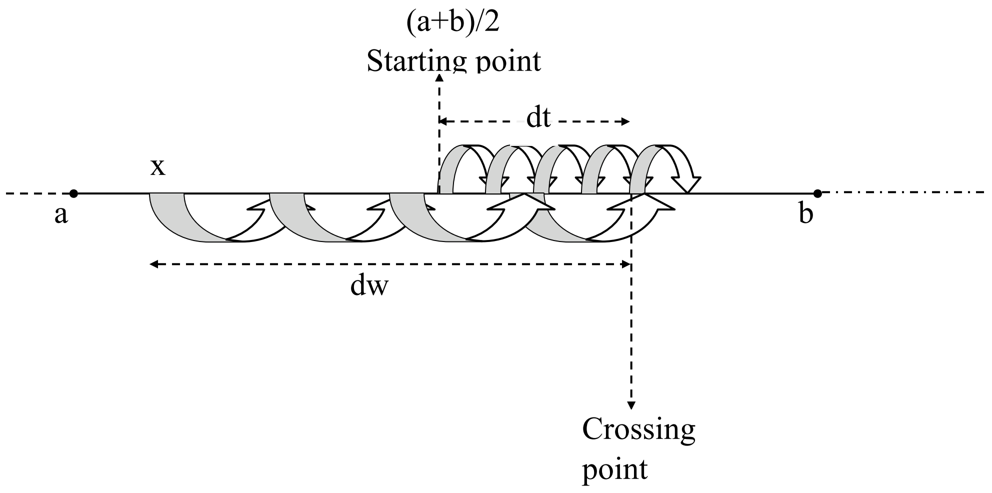
Lets try to calculate how much memory required for our problem.

It is necessary to store $\sqrt{p} \approx 10^{31}$ numbers in memory. Each of them is different length size between 1 to 62 digits. However most of them close to 62 digits. Therefore we can assume that length size of the numbers are about 60 digits.

60 digits≈ 200 bits ≈ 25 bytes.
Required Total memory is 10^{31} x 25 byte= 25×10^{22} GB !
It is also clear that why we didn't use this method.

Pollard's Lambda (Kangaroo)

This method is useful when the unknown logarithm is known to lie within a relatively small interval.



$x = \log_g y = (a+b)/2 + dt - dw$

The time complexity of this method $O(\sqrt{b-a})$ and space complexity is $O(log(b-a) \cdot log(n))$, where n is order of the group and a,b are limits of searching area. Since we don't know any small interval in which our keys would be inside it, we can't use this method.

Pollard's Rho Algorithm

Overview

The Pollard Rho Algorithm for solving the discrete log is as follows:

Given $\beta \equiv \alpha^e (mod p)$ we partition the set, $S = \langle\alpha\rangle$, into three subsets of more or less equal order[1]: S_1, S_2, S_3

Now we set $x_0 = 1$,[2] and define the following sequence of elements in $\mathbb{Z}_p^\times$

$$\begin{aligned} &\alpha x_i \text{ if } x_i \in S_1 \\ x_{i+1} = &x_i^2 \text{ if } x_i \in S_2 \\ &\beta x_i \text{ if } x_i \in S_3 \end{aligned}$$

now the values x_i,and x_{2i} are calculated for greater and greater i's until $x_i, = x_{2i}$ at this point the algorithm halts and we have

$$x_i = \alpha^{a_i}\beta^{b_i} = \alpha^{a_{2i}}\beta^{b_{2i}} = x_{2i}$$

so

$$\begin{aligned} \alpha^{a_i - a_{2i}} &= \beta^{b_{2i} - b_i} \\ \log_\alpha(\beta) &= \tfrac{a_i - a_{2i}}{b_{2i} - b_i} \end{aligned}$$

is a solution.

The Birthday Problem

The theory behind the Pollard Rho Algorithm is the same as that of the classic Birthday Problem/Paradox. If you have a set S of q "Birthdays" and you pick from it at random with replacement, the chance that you will have two of the same "Birthdays", that is a collision, after n choices is $1 - \prod_{i=1}^{n} \frac{q-(i-1)}{q}$ now in the case of the classic example of the Birthday Paradox this gives the result (which seems to surprise most people and thus "paradox") that after you have 22 people in the room there is a fifty percent chance .

That is

$$1 - \prod_{i=1}^{23} \frac{365-(i-1)}{365} = .\dot{5}073 \qquad *$$

[1] One must take care that S_2 does not contain 1. I used $x \in S_n$ if $x \equiv n(mod3)$

[2] in general x_0 must just be some known power of α

Now in the case of Pollard Rho it is assumed that our choice of x_i's that were described are sufficiently random and equally distributed so that we can expect a collision as dictated by (*) with q set equal to the order of 5 in the group $\mathbb{Z}_p^\times$. Now as soon as our first collision occurs, since our method of choosing x_i's is completely deterministic and based on the proceeding x_{i-1}, the sequence $\{x_i\}$ begins to repeat itself. Thus the probability that $\{x_i\}$ cycles through n or less values is, in the same manner as before given by (*) and thus also the probability that for some $m < n$ that $x_i = x_{2i}$.

The Index-Calculus Attack

Overview

The fastest attack on the discrete logarithm problem is the index-calculus algorithm with a probabilistic running time of $e^{(2+O(1))\sqrt{log(p)log(log(p))}}$. The simplest explanation for the attack is that it is based on the logarithmic identity, $log(a^m \cdot b^n) = m \cdot log(a) + n \cdot log(b)$. An interesting property of the algorithm is that once one logarithm has been computed, any logarithm may be computed with respect to the same base. This is because the index-calculus algorithm divides into two stages, with the first stage depending only on the base and not the logarithm in question.

How it Works

The attack has two stages: generating relations and computing logarithms using linear algebra. The first stage aims to find ways we can combine small primes in ways that are congruent to powers of the base of the logarithm. Then combining those relations will give the logarithm of all the small primes and then the logarithm of interest.

Having already been given a modulus, base value, and number to take to logarithm of, select a factor base. A factor base is a collection of primes, generally all of the primes below a boundary B. Numbers that factor over the factor base are called B-smooth.

With a fixed factor base, generate relations between the factor base and powers of the base value. The most basic approach to this problem is to take random powers of the base value (modulo a large number) and try to represent them as the product of primes in the factor base. If the number factors completely over the factor base then it is B-smooth and we have a relation. $c^k = \Pi p_i^{n_i}$

This relation also represents a vector with n_i in the ith location. Note that most of the spaces in this vector will be zero as the smooth number will not be divisible by most of the primes in the factor base. Test if the vector represented by the newly found relation is linearly independent from other relations found. Record it and the power it corresponds to if it is independent, otherwise discard it. Repeat until there are as many linearly independent relations and primes in the factor base.

The second stage of the algorithm is to combine the information from each individual relation into a meaningful whole. We can do this by arranging the relations and powers they correspond to in the form of a matrix equation:

$$\begin{pmatrix} n_{1,1} & n_{1,2} & ... & n_{1,m} \\ n_{2,1} & n_{2,2} & ... & n_{2,m} \\ ... & ... & ... & ... \\ n_{m,1} & n_{m,2} & ... & n_{m,m} \end{pmatrix} \cdot \begin{pmatrix} log_g p_1 \\ log_g p_2 \\ ... \\ log_g p_m \end{pmatrix} = \begin{pmatrix} k_1 \\ k_2 \\ ... \\ k_m \end{pmatrix}$$

And since all of the vectors generated by the relations are independent, the square matrix on the left (call it A) is invertible. Giving the equation:

$$\begin{pmatrix} log_g p_1 \\ log_g p_2 \\ ... \\ log_g p_m \end{pmatrix} = A^{-1} \cdot \begin{pmatrix} k_1 \\ k_2 \\ ... \\ k_m \end{pmatrix}$$

And by performing one matrix multiplication we have the logarithm of all the primes in the factor base. One note of caution, the exponents in the matrices are modulo the order of the base element, not the original modulus of the finite field.

Finally, find a power of the base, g, such that (with unknown a) g^{a+j} is B-smooth. Then the logarithm $log_g(g^{a+b}) = a + b = log(\Pi p_i^{n_i}) = \Sigma n_i log_g(p_i)$. And we know what b is, so the unknown $a = (-b) + \Sigma n_i log_g(p_i)$

A (Simple) Example

In the finite field modulo 17, let the base element, g, be 3 and the bound for the factor base be 5. Compute the logarithm of 6.

$3^1 = 3$
$3^3 = 10 = 2 \cdot 5$
$3^4 = 13$ (not smooth)
$3^5 = 5$
$3^8 = 16 = 2^4$

From the relations given by 3^3 and 3^5 we get the logarithm of 2. Knowing the logarithm of 3 is 1, then $log(6) = log(3) + log(2) = 1 + 14 = 15$.

Our Struggle with the Index Calculus

The first hurdle faced in implementing the index-calculus algorithm is the sheer size of the numbers involved. The modulus for our group had sixty-two digits and wouldn't fit in any standard data type for the C language. Luckily, a very fast multiple-precision library was written by the Free Software Foundation called the GNU Multiple Precision Library, or GMPLib (http://gmplib.org/).

Similarly, creating a matrix of relations for a large problem could require billions of entries. Tracking each entry individually would require gigabytes of memory. Fortunately, someone came up with the idea of a sparse matrix. A sparse matrix is made up of mostly zeroes (like the relations), and the data type uses memory only to track the non-zero entries and saving a lot of memory. ScaLAPACK (http://www.netlib.org/scalapack/) and the Optimized Sparse Kernel Interface (http://bebop.cs.berkeley.edu/oski/) provide efficient C libraries for dealing with sparse matrices.

The first problem we faced that had no obvious solution was how to deal with the fact that our base value was not a primitive root in the finite field. Given the large prime, p, the number of elements of the group, (p - 1), has factors of 2 and (p - 1)/2. The order of our base element is (p - 1)/2, which is a prime.

This means that five generates a cyclic group of prime order through multiplication with itself. Prime order cyclic groups have the characteristic that every non-identity element generates the entire group. Thus every element that is a power of five has order (p - 1)/2.

Unfortunately, this also means that half of the finite field is not in the image five. Not only are there large numbers that have no valid logarithm, but small primes in the factor base may not have a logarithm. Including these small primes in the factor base would preclude the index-calculus algorithm from working. Not having a logarithm, these primes would not be separable in the matrix of relations. There do not exist linearly independent relations for these primes, so the relations finding step of the algorithm would never terminate.

There is a relatively quick test to determine if a number is in the image of five. Since all powers of five are of the form 5^k, then raising that number to the (p - 1)/2 power gives $(5^k)^{(p-1)/2} = (5^{(p-1)/2})^k = 1^k = 1$. Numbers not in the image of five are of the form $(-1)5^k$, raising these to the (p - 1)/2 power gives $(-1)^{(p-1)/2}(5^k)^{(p-1)/2} = (-1)(5^{(p-1)/2})^k = (-1)(1^k) = -1$. Using this fact when constructing the factor base, we can ensure that we only include primes with valid logarithms.

The most trying problem in implementing the index calculus algorithm is finding smooth numbers quickly. Without the fast generation of smooth numbers there are no relations to compute the logarithms of primes in the factor base and the entire algorithm fails. There are a few tricks you can use to speed up the generation of smooth numbers.

The best tip for generating smooth numbers is not to use random powers of the base element. These numbers will be about the same size as the modulus, which is to say, they will be very large. Instead, if we set a constant h to the first integer larger than $\sqrt{p}$ then $h^2 = p + j \equiv j \mod p$ for some constant j. We can then compute all the values of $(h+a)(h+b) = h^2 + (a+b)h + ab \equiv j + (a+b)h + ab$, which are about the same size as $\sqrt{p}$. Being much smaller, there are many fewer possible primes that could be factors of these numbers and they are more likely to be B-smooth. The drawback to this method is that the $(h+a)$ and $(h+b)$ have to be added to the factor base (and thus be in the image of five). The upside is that for $0 < a, b < n$ the equation generates $(n^2+n)/2$ likely smooth numbers, thus giving many possible relations to determine the logarithm of these numbers.

Even using the $(h+a)(h+b)$ method, we found only about one smooth number in every hundred thousand tested. Selecting those numbers quickly could be greatly sped up by implementing a polynomial sieve. The polynomial sieve would take a function, say $(h+a)(h+x)$ for a fixed a, and given one smooth x output many more. The principle behind the sieve is that if, $\mod p_i$ (p_i in the factor base), a function has a root r then the function has a root of $r + n \cdot p_i$ not modulo p_i. This creates a set of equivalence classes for roots of the function and once once root (a smooth number) is found, we can generate many more quickly and cheaply by noting that if $f(x) = 0 \mod p_i$ then $f(x + p_i) = 0 \mod p_i$.

Bibliography

R. Crandall, C. Pomerance, "Prime Numbers a Computational Perspective", Second edition. Springer, New York, 2005

A. Menezes, P. van Oorschot, S. Vanstone, "Handbook of Applied Cryptography", CRC Press, 1996

P. Montgomery, "A Block Lanczos Algorithm for Finding Dependencies over GF(2)"

A. M. Odlyzko, "Discrete Logarithms in Finite Fields and their Cryptographic Significance", AT&T Bell Laboratories, Murray Hill, New Jersey 07974.

C. Pomerance, "Elementary Thoughts on Discrete Logarithms", to appear in the proceedings of an MSRI workshop, J. Buhler and P. Stevenhagen, eds.

C. Pomerance, "Smooth Numbers and the Quadratic Sieve", to appear in the proceedings of an MSRI workshop, J. Buhler and P. Stevenhagen, eds

O. Schirokauer, D. Weber, and T. Denny, "Discrete logarithms: The effectiveness of the index calculus method", pp. 337--362 in Algorithmic Number Theory: Second Intern. Symp., ANTS-II, H. Cohen, ed., Lecture Notes in Math. #1122, Springer, 1996. http://citeseer.ist.psu.edu/article/schirokauer96discrete.html

T. Stegers,"Aspects of the Discrete Logarithm", Thesis fulfilling requirements for the degree of Bachelor of Science, Technische Universität Darnstadt

C. Studholme, "The Discrete Log Problem", Thesis fulfilling requirements for the degree of Doctor of Philosophy in Mathematics, University of Toronto

W. Trappe, L. Washington, "Introduction to Cryptography with Coding Theory", Second edition, Prentice Hall, Upper Saddle River NJ 07458 USA. 2002

The GMP Team, "The GNU Multiple Precision Arithmetic Library", Free Software Foundation, Inc., http://gmplib.org/

Pohlig-Hellman algorithm, In Wikipedia, The Free Encyclopedia. Retrieved November 2, 2007, from http://en.wikipedia.org/wiki/Pohlig-Hellman_algorithm

Pollard's rho algorithm for logarithms, In Wikipedia, The Free Encyclopedia. Retrieved November 2, 2007, from http://en.wikipedia.org/wiki/Pollard's_rho_algorithm_for_logarithms

Index calculus algorithm, In Wikipedia, The Free Encyclopedia. Retrieved November 2, 2007, from http://en.wikipedia.org/wiki/index_calculus

Baby-step Giant-step, In Wikipedia, The Free Encyclopedia. Retrieved November 2, 2007, from http://en.wikipedia.org/wiki/Baby-step_giant-step
B Kernighan, D. Ritchie, "The C Programming Language", Second edition, Prentice Hall, Inc., 1988