# DISCRETE LOGARITHMS

Karthik Kuber

# The Discrete Logarithm (DL) Problem:

Let G be a finite cyclic group with n elements. Let b be a generator of G, with neutral element 1 - every element g of G can be written in the form

$$g = b^k \qquad \text{....(1)}$$

for some integer k.

*k is called the discrete logarithm of g to the base b. The DL problem is to find the least such non-negative integer k.*

# The DL Problem generalized:

*For any group H, which is not necessarily cyclic, given g and b, is there a value of k satisfying the above equation? If so, what is its least non-negative value?*

In cryptographic applications, however, k always exists and it only needs to be found by the attacker. Hence, we shall limit our discussion to the specific DL problem, viz. in the case of only cyclic groups, for which k always exists.

# Finding the discrete logarithm:

1. Enumeration
2. Shanks Baby-step Giant-step Algorithm
3. The Pollard ρ-Algorithm
4. The Pollard λ-Algorithm
5. The Pohlig-Hellman Algorithm
6. Index Calculation Algorithm
7. Others (Variants of Index Calculation Algorithms like Number Field Sieve, Function Field Sieve, etc.)

# Enumeration

The most fundamental of algorithms – Involves testing values of k = 0,1,2,3,.... and checking if it satisfies equation (1). Although its space complexity is very low requiring space for only 3 elements, this is clearly a naive algorithm.

Example: $3^k = 4 \bmod 7$

k=0, $3^0 = 1 = 1 \bmod 7$

k=1, $3^1 = 3 = 3 \bmod 7$

k=2, $3^2 = 9 = 2 \bmod 7$

k=3, $3^3 = 27 = 6 \bmod 7$

k=4, $3^4 = 81 = 4 \bmod 7$

# Shanks Baby-step Giant-step Algorithm (Modified version of the enumeration algorithm)

$$g = b^k \text{ ....let } k=qm + r$$
$$g = b^{qm+r} \rightarrow g(b^{-m})^q = b^r$$

- Calculate the set of all values of $b^r$ for $0 \leq r < \text{ceil}(\sqrt{n})$ and stores them in a hash table
- Check each value of $g(b^{-m})^q$ for all values of q, $0 \leq q < \text{ceil}(\sqrt{n})$ against the stored values in the hash table. If it matches, k = qm+r

Space complexity = $\sqrt{n}$

# The Pollard ρ-Algorithm

To find $\alpha^{\gamma}=\beta \mod N$, compute integers a,b,A,B such that $\alpha^{a}\beta^{b} = \alpha^{A}\beta^{B} \pmod n$

To find a,b,A and B, this algorithm uses Floyd's cycle-finding algorithm to find a cycle in the sequence

$$x_i = \alpha^{ai}\beta^{bi}$$

Divide G into 3 subsets of approx equal size.
If $x_i$ is in $G_0$, double both x and y;
if $x_i$ is in $G_1$, increment a;
if $x_i$ is in $G_2$, increment b.
Calculate the same with $A_i$ and $B_i$. Store it in $X_i$. Break when $x_i = X_i$. This gives the values of a,b,A,B.
Then $(B - b)\gamma = (a - A) \pmod n$

# Example:

| i | x | a | b | X | A | B |
|---|---|---|---|---|---|---|
| 1 | 2 | 1 | 0 | 10 | 1 | 1 |
| 2 | 10 | 1 | 1 | 100 | 2 | 2 |
| 3 | 20 | 2 | 1 | 1000 | 3 | 3 |
| 4 | 100 | 2 | 2 | 425 | 8 | 6 |
| 5 | 200 | 3 | 2 | 436 | 16 | 14 |
| 6 | 1000 | 3 | 3 | 284 | 17 | 15 |
| 7 | 981 | 4 | 3 | 986 | 17 | 17 |
| 8 | 425 | 8 | 6 | 194 | 17 | 19 |
| ... | ... | ... | ... | ... | ... | ... |
| 48 | 224 | 680 | 376 | 86 | 299 | 412 |
| 49 | 101 | 680 | 377 | 860 | 300 | 413 |
| 50 | 505 | 680 | 378 | 101 | 300 | 415 |
| 51 | 1010 | 681 | 378 | 1010 | 301 | 416 |

$2^{681}5^{378} = 1010 = 2^{301}5^{416} \pmod{1019}$ and

$(416 - 378)\gamma = 681 - 301 \pmod{1018}$, for which $\gamma_1 = 10$ is a solution.

Further, since n = 1018 is not prime, another solution exists: $\gamma_2 = 519$, for which $2^{519} = 1014 = -5 \pmod{1019}$

{Source: http://en.wikipedia.org/wiki/Pollard%27s_rho_algorithm_for_logarithms}

# The Pollard λ-Algorithm

Search in a subset {a,...,b} of G with $x_{i+1} = x_i\alpha^{f(xi)}$ where f is a pseudorandom map, and the initial state, $x_0 = \alpha^b$, and then compute a value D which is the sum of all values of $f(x_i)$ from 0 to N-1. Now, do the same with a set f values y, with initial value $y_0 = \beta$. When $y_j = x_N$ for some j, we have $x = b+d-d_j \ (\mathrm{mod} p)$

# The Pohlig-Hellman Algorithm

For $g = b^k \bmod p$,

$f(p) = p_1 p_2 ... p_n$ (Totient Function),

$k = a_1 p_1 + c_1$, choose a $c_1$ such that

$$g^{f(p)/p1} = (b^k)^{f(p)/p1} \bmod p$$
$$= (b^{f(p)})^{a1} . b^{c1 f(p)/p1} \bmod p$$
$$= b^{(f(p)/p1)c1} \bmod p$$

The above steps are repeated for $p_2, p_3 ... p_n$

So we have n congruences and these can be solved for k using the Chinese Remainder Theorem

Example: $5^x = 3 \bmod 2017$

$n = 2016 = 2^5 * 3^2 * 7$

$(5^{9*7})^{x(2)} = 3^{9*7} \bmod 2017$

or $500^{x(2)} = 913 \bmod 2017$

$x(2) = 6$

Similarly, x(3) and x(7) are calculated.

So, $x = 6 \bmod 2^5$

$x = 4 \bmod 3^2$

$x = 1 \bmod 7$

Solving, $x = 1030$

{Source: Johannes A. Buchmann, 2000, Introduction to Cryptography, Springer-Verlag New York, Inc., USA}

# The Index Calculus Algorithm

Best known for groups $Z^*_m$

Uses factor base as input (usually -1 or 2 and primes from 2). For $g = b^k$,

For m = 0,1,2,... factor gm such that

$g^m = (-1)^{e0}2^{e1}3^{e2}....p_r^{er}$

Store m and all $e_i$ as a vector(relation)

If this relation is linearly independent to other relations, add this relation to relations, until there are at least r relations.

Form a matrix with these relations as rows.

Obtain a reduced echelon form of this matrix with elements in the last

column being discrete logs of the factor bases.
Then for n = 0,1,2,... factor
$gb^n = (-1)^{f0}2^{f1}2^{f2}....p_r^{fr}$
$k = f_0\log_b(-1) + f_1\log_b 2 +...+ f_r\log_b p_r - n$

This has sub-exponential running time
$L_p[1/2,c+o(1)]$

Example:

Let p = 229. The element b=6 is a generator of $Z_{229}$ of order n=228. Consider g=13. Then $\log_6 13$ is computed as follows:

1. The factor base is chosen to be the first 5 primes: S = {2, 3, 5, 7, 11}

2. The following 6 relations are obtained:

$6^{100} \bmod 229 = 180 = 2^2 . 3^2 . 5$

$6^{18} \bmod 229 = 176 = 2^4 . 11$

$6^{12} \bmod 229 = 165 = 3.5.11$

$6^{62} \bmod 229 = 154 = 2.7.11$

$6^{143} \bmod 229 = 198 = 2.3^2 . 11$

$6^{206} \bmod 229 = 210 = 2.3.5.7$

These relations yield the following equations:

$100 = 2 \log_6 2 + 2\log_6 3 + \log_6 5 \pmod{228}$

$18 = 4 \log_6 2 + \log_6 11 \pmod{228}$

$12 = \log_6 3 + \log_6 5 + \log_6 11 \pmod{228}$

$62 = \log_6 2 + \log_6 7 + \log_6 11 \pmod{228}$

$143 = \log_6 2 + 2\log_6 3 + \log_6 11 \pmod{228}$

$206 = \log_6 2 + \log_6 3 + \log_6 5 + \log_6 7 \pmod{228}$

3. Solving the linear system of 6 equations in 5 unknowns ($x_i = \log_6 p_i$) yields:

$\log_6 2=21, \log_6 3=208, \log_6 5=98, \log_6 7=107, \log_6 11=162$

4. Finally, suppose n = 77 is selected.
Since $gb^n = 13*6^{77} \bmod 229 = 147 = 3*7^2$, it follows:
$\log_6 13 = (\log_6 3+2\log_6 7-77) \bmod 228 = 117$

{Source: Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone, Handbook of Applied Cryptography, Chapter 3, 2001.}

# Others

The fastest (currently) index calculus algorithm is the number field sieve, which has running time of $L_p[1/3,(64/9)^{1/3}]$

Other efficient factoring algorithms also have DL variants. Hence this problem is very much similar to algorithms based on the difficulty of integer factorization. However, some alternatives are DL problems on elliptic curves or in fields.

# References:

1. Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone, Handbook of Applied Cryptography, 2001

2. Source: Johannes A. Buchmann, 2000, Introduction to Cryptography, Springer-Verlag New York, Inc., USA

3. Wikipedia articles:

   http://en.wikipedia.org/wiki/Discrete_logarithm
   http://en.wikipedia.org/wiki/Baby-step_giant-step
   http://en.wikipedia.org/wiki/Pollard%27s_rho_algorithm_for_logarithms
   http://en.wikipedia.org/wiki/Pollard%27s_lambda_algorithm
   http://en.wikipedia.org/wiki/Pohlig-Hellman_algorithm
   http://en.wikipedia.org/wiki/Index_calculus_algorithm