Algorithms for Computing Discrete Logarithms

AK Bhateja

Algorithms for solving Discrete log problem

- Exhaustive search
- Baby-step giant-step algorithm
- Pollard’s rho algorithm for logarithms
- Pohlig-Hellman algorithm
- Index-calculus algorithm

Exhaustive search

- Successively compute $\alpha, \alpha^2, \ldots, \alpha^n$
- This method takes $O(n)$ multiplications, where n is the order of α
- Inefficient if n is large

Baby-step giant-step algorithm

- Developed by Shanks
- Let G be a cyclic group of order n, α is a generator of G.
- $m = \lceil \sqrt{n} \rceil$
- If $\beta = \alpha^x$, then write $x = qm + r$, where $0 \leq q, r < m$.
- Therefore $\alpha^x = \alpha^{qm} \alpha^r$ i.e. $\beta(\alpha^{-m})^q = \alpha^r$
- The algorithm
  - **Baby Step**: If for some r, $\beta \alpha^{-r} = 1$, then $\beta = \alpha^r$
  - **Giant step**: Find $\beta \alpha^{-mq}$; $q = 0, 1, 2, \ldots$

    Compare this with α^r ; $r = 0, 1, 2, \ldots$ till $\beta \alpha^{-mq} = \alpha^r$

Baby-step giant-step algorithm

Given: a generator α of a cyclic group of order n and an element β

Set $m \leftarrow \lceil \sqrt{n} \rceil$

for $r = 0$ to $m - 1$

compute α^{r} (mod n) and store the pair (j, α^{r}) in a table.

easily searchable on the first coordinate

compute α^{-m}

$\gamma \leftarrow \beta$

for $q = 0$ to $m - 1$

if $\gamma = \alpha^{r}$ for some r in the table

return $qm + r$

else $\gamma \leftarrow \gamma \cdot \alpha^{-m}$ mod n

Time complexity of Baby-step giant-step algorithm

- Memory (storage) requirement O($\sqrt{n}$)
- Construction of table: It requires O($\sqrt{n}$) multiplications
- Sorting the table: Sort the table by second component, it requires O($\sqrt{n}$ lg n) comparisons
  - Alternatively, use hashing on the second component to store the entries in a hash table; placing an entry, and searching for an entry in the table takes constant time.
- For all q where $0 \leq q < m$, it is required to sea33333rch α^{r}, which is equal to γ. It requires O($\sqrt{n}$) multiplications and O($\sqrt{n}$) table look-ups
- The running time of the algorithm is O($\sqrt{n}$) multiplications.

Example: Let $p = 113$, $\alpha = 3$ and $\beta = 57$

$m = \lceil \sqrt{112} \rceil = 11$

r	0	1	2	3	4	5	6	7	8	9	10
3^r mod 113	1	3	9	27	81	17	51	40	7	21	63

Find $\alpha^{-1} = 3^{-1} \bmod 113 = 38$, $\alpha^{-m} = 58$

$\gamma = \beta\alpha^{-mq} \bmod 113$ for $q = 0, 1, 2, \ldots$ is computed until a value in the second row of the table is obtained.

q	0	1	2	3	4	5	6	7	8	9
$\gamma = 57 \cdot 58^q \bmod 113$	57	29	100	37	112	55	26	39	2	3

Since $\beta\alpha^{-mq} \bmod 113 = 3 = \alpha^1$, $qm + r = 9 \times 11 + 1 = 100$

$\therefore \log_3 57 = 100$

Pollard’s rho algorithm for Discrete Logarithms

- Pollard proposed an elegant algorithm in 1978
- Pollard’s rho algorithm for computing discrete logarithms is a randomized algorithm
- Its expected running time is same as the baby-step giant-step algorithm i.e. $O(\sqrt{p})$
- It requires a negligible amount of storage.
- Therefore, it is far preferable to baby-step giant-step algorithm for problems of practical interest.

Pollard's rho algorithm for Discrete Logarithms

- Let G be a cyclic group of order n (prime), with generator α, $\beta \in G$
- Find integers a, b, A, B s.t. $\alpha^{a} \beta^{b} = \alpha^{A} \beta^{B}$
- The solution of equation $(B - b)\, x = (A - a)$ is $x = \log_{\alpha} \beta$
- For finding a, b, A, B , Floyd's cycle-finding algorithm can be used to find a cycle in the sequence $x_i = \alpha^{a_i}\beta^{b_i}$. i.e. to find two group elements x_i and x_{2i} such that $x_i = x_{2i}$.
- Hence $\alpha^{a_i}\beta^{b_i} = \alpha^{a_{2i}}\beta^{b_{2i}} \Rightarrow \beta^{b_i - b_{2i}} = \alpha^{a_{2i} - a_i}$
- Therefore, by taking log both sides $(b_i - b_{2i})\log_{\alpha}\beta = (a_i - a_{2i}) \bmod n$ provided $b_i \neq b_{2i} \bmod n$. (note: $b_i \equiv b_{2i} \bmod n$ occurs with probability ≈ 0
- Solution to this equation can be easily obtained using Euclidean algorithm.

Pollard's rho algorithm for Discrete Logarithms

- Divide the group G into three pairwise disjoint subsets S_0, S_1 and S_2 with $G = S_0 \cup S_1 \cup S_2$ roughly equal in size.
- Let $f : G \to G$ defined by

$$x_{i+1} = f(x_i) = \begin{cases} \beta \cdot x_i & if\ x_i \in S_0 \\ x_i^2 & if\ x_i \in S_1 \\ \alpha \cdot x_i & if\ x_i \in S_2 \end{cases}$$

The initial term is $x_0 = \alpha^{a_0}\beta^{b_0}$ for random values of a_0 & b_0, $1 \notin S_2$

- This sequence of group elements in turn defines two sequences of integers a_0, a_1, a_2,... and b_0, b_1, b_2,... satisfying $x_i = \alpha^{a_i}\beta^{b_i} \bmod p$ for $i \geq 0$.

- For $i \geq 0$: $a_0 = 0$, $b_0 = 0$ (both may be randomly selected elements of Z_p)

  define

$$a_{i+1} = \begin{cases} a_i & if\ x_i \in S_0 \\ 2a_i \bmod n & if\ x_i \in S_1 \\ a_i + 1 \bmod n & if\ x_i \in S_2 \end{cases}$$

and

$$b_{i+1} = \begin{cases} b_i + 1 \bmod n & if\ x_i \in S_0 \\ 2b_i \bmod n & if\ x_i \in S_1 \\ b_i & if\ x_i \in S_2 \end{cases}$$

- Find two group elements x_i and x_{2i} such that $x_i = x_{2i}$

Pollard's rho algorithm for computing discrete logarithms

Given: a generator α of a cyclic group G of order n, and an element $\beta \in G$

Set $x_0 = 1$, $a_0 = 0$, $b_0 = 0$

for $i = 1, 2, \ldots$

    find x_i, a_i, b_i, and x_{2i}, a_{2i}, b_{2i}

    if $x_i = x_{2i}$

        set $r = b_i - b_{2i} \bmod n$.

        if $r = 0$ then terminate the algorithm with failure

        else

            return $x = r^{-1}(a_{2i} - a_i) \bmod n$

- In rare case it terminates with failure, the procedure can be repeated by selecting random integers a_0, b_0 in the interval $[1, n-1]$, and $x_0 = \alpha^{a_0}\beta^{b_0}$

Pollard's rho algorithm: Example

Let α = 5 be a generator of a cyclic group Z^*_{2017}
$\beta = 1736$

Divide the set $S = \{1, 2, \dots, 2016\}$ into 3 sets

$S_i = \{a \mid a \in S \;\&\; a = i \bmod 3\}$ for $i =$ 0, 1, 2

Initialization: Let $a_0 = 27$, $b_0 = 0$,

$x_0 = \alpha^{a_0}\beta^{b_0} = 5^{27} \bmod 2017 \equiv 710$

i	x_i	S_0	S_1	S_2	a_i	b_i
0	$x_0 = 710$			710	27	0
1	$x_1 = \alpha \cdot x_0 \bmod 2017 = 1533$	1533			28	0
2	$x_2 = \beta \cdot x_1 \bmod 2017 = 865$		865		28	1
3	$x_3 = x_2^2 \bmod 2017 = 1935$	1935			56	2
4	$x_4 = \beta \cdot x_3 \bmod 2017 = 855$	855			56	3
5	$x_5 = \beta \cdot x_4 \bmod 2017 = 1785$	1785			56	4
6	$x_6 = \beta \cdot x_5 \bmod 2017 = 648$	648			56	5
7	$x_7 = \beta \cdot x_6 \bmod 2017 = 1459$		1459		56	6
8	$x_8 = x_7^2 \bmod 2017 = 746$			746	112	12
9	$x_9 = \alpha \cdot x_8 \bmod 2017 = 1713$	1713			113	12
10	$x_{10} = \beta \cdot x_9 \bmod 2017 = 710$			710	113	13

Example Pollard Rho algorithm

- Since 710 appears twice in the 0th and 10th rows, we have

$$\alpha^{27}\beta^{0} = \alpha^{113}\beta^{13} \Rightarrow \beta^{13-0} = \alpha^{27-113}$$

$$\Rightarrow (\alpha^{x})^{13} = \alpha^{27-113}$$

$$\Rightarrow \alpha^{13x} = \alpha^{27-113}$$

$$\Rightarrow 13x = (27 - 113) \bmod 2016$$

$$\Rightarrow x = 13^{-1} \times 1930 \bmod 2016$$

$$= 1861 \times 1930 \bmod 2016$$

$$= 1234$$

Pohlig-Hellman (Silver Pohling-Hellman) algorithm

- Discovered by Roland Silver, but first published by Stephen Pohlig and Martin Hellman.
- It applies to groups whose order is a primes power
- Consider a finite cyclic abelian group $Z_p{}^*$, with order p -1.

$$p - 1 = p_1^{e_1} \cdot p_2^{e_2} \cdots\ p_k^{e_k}$$

- Idea: compute x mod $p_i^{e_i}$ for each i, $1 \le i \le k$ then compute x mod $(p - 1)$ using Chinese remainder theorem

The Pohlig-Hellman Algorithm

Consider prime p, α is a generator of Z_p^* and $\beta \in Z_p^*$.

Goal: to determine $x = \log_\alpha \beta \bmod (p - 1)$

Let $p - 1 = p_1^{e_1} \cdot p_2^{e_2} \cdots p_k^{e_k}$ where the p_i's are distinct primes

Idea: compute $x \bmod p_i^{e_i}$ for each i, $1 \leq i \leq k$ then compute $x \bmod (p - 1)$ using Chinese remainder theorem

Suppose that $q = p_i$ and $e = e_i$,

How to compute the value $a = x \pmod{q^e}$?

Express a in radix q representation as

$$a = x_0 + x_1 q + \ldots + x_{e-1} q^{e-1} \; ; \quad 0 \leq x_i \leq q^e$$

$$a = x \pmod{q^e} \Rightarrow x = a + q^e s \text{ for some integer } s.$$

$$\therefore \quad x = x_0 + x_1 q + \ldots + x_{e-1} q^{e-1} + s q^e$$

Step 1: to find x_0. x_0 can be computed using the fact

Fact: $\beta^{\frac{p-1}{q}} \equiv \alpha^{\frac{x_0(p-1)}{q}} \bmod p$ … (1)

Proof: $\beta^{\frac{p-1}{q}} \equiv (\alpha^x)^{\frac{(p-1)}{q}} \bmod p$

$$\equiv \left(\alpha^{x_0+x_1 q+\cdots+x_{e-1}+sq^e}\right)^{\frac{(p-1)}{q}} \bmod p$$

$$\equiv (\alpha^{x_0+Kq})^{\frac{(p-1)}{q}} \bmod p \quad \text{where K is an integer}$$

$$\equiv \alpha^{x_0\left(\frac{p-1}{q}\right)}\alpha^{K(p-1)} \bmod p$$

$$\equiv \alpha^{x_0\left(\frac{p-1}{q}\right)} \bmod p \quad \text{because } \alpha^{p-1} \equiv 1 \bmod p$$

Algorithm to compute x_0

Compute $\beta^{\frac{p-1}{q}}$

If $\beta^{\frac{p-1}{q}} \equiv 1 \bmod p$ then $x_0 = 0$

Otherwise successively compute

$$\gamma = \alpha^{\frac{p-1}{q}} \bmod p,\ \gamma^2 \bmod p, \ldots$$

until $\gamma^i \equiv \beta^{\frac{p-1}{q}} \bmod p$

return ($x_0 = i$)

step 2: to compute $x_1, x_2, \ldots, x_{e-1}$ if $e > 1$

Define $\beta_0 = \beta$, and

$$\beta_j = \beta\alpha^{-(x_0+x_1 q+\cdots+x_{j-1}q^{j-1})} \bmod p, \text{ for } 0 \le j \le e - 1$$

Consider the generalization of equation (1)

$$(\beta_j)^{\frac{p-1}{q^{j+1}}} \equiv \alpha^{\frac{x_j(p-1)}{q}} \bmod p \qquad \ldots \ (2)$$

Proof: When $j = 0$, this equation reduces to equation (1)

$$\left(\beta_j\right)^{\frac{p-1}{q^{j+1}}} \equiv \left(\beta\alpha^{-\left(x_0+x_1q+\cdots+x_{j-1}q^{j-1}\right)}\right)^{\frac{p-1}{q^{j+1}}} \bmod p$$

$$\equiv \left(\alpha^{x-\left(x_0+x_1q+\cdots+x_{j-1}q^{j-1}\right)}\right)^{\frac{p-1}{q^{j+1}}} \bmod p$$

$$\equiv \left(\alpha^{x_jq^j+\cdots+x_{e-1}q^{e-1}+sq^e}\right)^{\frac{p-1}{q^{j+1}}} \bmod p$$

$$\equiv \left(\alpha^{x_jq^j+Kq^{j+1}}\right)^{\frac{p-1}{q^{j+1}}} \bmod p \text{ where } K \text{ is an integer}$$

$$\equiv \alpha^{\frac{x_j(p-1)}{q}}\left(\alpha^{p-1}\right)^K \bmod p$$

$$\equiv \alpha^{\frac{x_j(p-1)}{q}} \bmod p$$

Hence we can compute x_j using this equation

Computation of $log_{\alpha}\beta \bmod p_i^{e_i}$

Since $\beta_j = \beta\alpha^{-(x_0+x_1q+\cdots+x_{j-1}q^{j-1})} \bmod p$,

for $0 \le j \le e - 1$

$\therefore\ \beta_{j+1} = \beta_j\alpha^{-x_j q^j} \bmod p$ … (3)

Now compute x_0, β_1 , x_1, β_2 , … , , β_{e-1} , x_{e-1} by alternatively applying equation (2) and (3).

$$log_{\alpha}\beta \bmod p_i^{e_i} = \sum_{i=0}^{e-1} x_i q^i$$

Algorithm: Pohlig-Hellman to compute $\log_\alpha \beta \pmod{q^e}$

- Compute $\gamma_i = \alpha^{(p-1)i/q} \bmod p$ for $0 \le i \le q-1$
- set $j = 0$ and $\beta_j = \beta$
- while $j \le e-1$

  compute $\delta = (\beta_j)^{\frac{(p-1)}{q^{j+1}}} \bmod p$

  find i such that $\delta = \gamma_i$

  $x_j = i$

  $\beta_j = \beta\alpha^{-x_j q^j} \bmod p$

  $j = j + 1$

Example: $p = 29$, $\alpha = 2$, $\beta = 18$

$p - 1 = 28 = 2^2 \cdot 7$

$q = 2$, $e = 2$;

$\gamma_0 = 1$

$\gamma_1 = \alpha^{(p-1)/q} \bmod p = 2^{14} \bmod 29 \equiv 28$

$\delta = \beta^{28/2} \bmod 29 = 18^{14} \bmod 29 = 28$

Hence $x_0 = 1$.

Compute $\beta_1 = \beta_0 \alpha^{-1} \bmod 29 = 9$

And $(\beta_1)^{28/4} = 9^7 \bmod 29 = 28$; since $\gamma_1 = 28$

Therefore $x_1 = 1$; Hence $x = 3 \bmod 4$

Similarly for $q = 7$, $x = 4 \bmod 7$

Using CRT $x = 11 \bmod 28$. i.e. $\log_2 18$ in Z_{29} is 11.

References

- Cryptography: Theory and Practice by Douglas R. Stinson
- The PohligHellman Algorithm by D.R. Stinson

  http://anh.cs.luc.edu/331/notes/PohligHellmanp_k2p.pdf

Index-calculus algorithm

- The index-calculus algorithm is the most powerful method for computing discrete logarithms.
- This is a subexponential-time algorithm.
- The index-calculus algorithm requires the selection of a relatively small subset S of elements of G, called the factor base, in such a way that a significant fraction of elements of G can be efficiently expressed as products of elements from S.

Computing Discrete Logarithms modulo *p*

- Initial processing stage
  - Computes the discrete logarithms of a set *S* of elements of Z_p^*
  - i.e. find a number of equations in the logarithms of *S* and solve over modulo $p - 1$.
- Final processing stage
  - Logarithm of any other may be found relatively quickly utilizing the logarithms of the elements of *S*.

Index Calculus Algorithm for DL in Cyclic Group

- Given: α generator of cyclic group Z_p, $\beta \in Z_p$.

1. Select factor base S: Consider a set of first t primes
   $S = \{p_1, p_2, \ldots, p_t\}$
2. Collect linear relations involving logarithms of elements in S
   - 2.1 Select a random integer $k \in [0, p-1]$, and compute α^k
   - 2.2 factorize α^k as a product of elements in S:

$$\alpha^k = \prod_{i=1}^{t} {p_i}^{e_i}$$

If successful, take logarithms of both sides of equation to obtain a linear relation

$$k = \sum_{i=1}^{t} e_i \, log_\alpha(p_i) \; mod \; (p-1)$$

   - 2.3 Repeat until $t + c$ relations of the form are obtained

Index Calculus Algorithm for DL in Cyclic Group

3. Find the logarithms of elements in S: Solve the linear congruent system of $t + c$ equations (in t unknowns) collected in step 2 to obtain the values of $\log_\alpha p_i$; $1 \le i \le t$.
4. Compute $\log_\alpha \beta$
   4.1 Select a random integer k, $0 \le k \le n - 1$, & compute $\beta \cdot \alpha^k \bmod p$
   4.2 factorize $\beta \cdot \alpha^k \bmod p$ in S

$$\beta \alpha^k = \prod_{i=1}^{t} {p_i}^{f_i}$$

   If the attempt is unsuccessful then repeat step 4.1.
   Otherwise, taking logarithms of both sides & compute $\log_\alpha \beta$

$$log_\alpha(\beta) \equiv \sum_{i=1}^{t} f_i \, log_\alpha(p_i) \; mod \; (p-1) - k$$

- Example: Index calculus for DL

Let $p = 229$ and its generator is $\alpha = 6$, $\beta = 13$

Factor base $S = \{2, 3, 5, 7, 11\}$

Generation of relations: Pick random numbers

$k = 100$	$6^{100} \bmod 229 \equiv 180 = 2^2 \times 3^2 \times 5$
$k = 18$	$6^{18} \bmod 229 \equiv 176 = 2^4 \times 11$
$k = 12$	$6^{12} \bmod 229 \equiv 165 = 3 \times 5 \times 11$
$k = 62$	$6^{62} \bmod 229 \equiv 154 = 2 \times 7 \times 11$
$k = 143$	$6^{143} \bmod 229 \equiv 198 = 2 \times 3^2 \times 11$
$k = 206$	$6^{206} \bmod 229 \equiv 210 = 2 \times 3 \times 5 \times 7$

Taking log with base α, both sides

Taking log with base α, both sides

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

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

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

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

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

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

- Solving the linear system equations:

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

- Computing the value:

Let $k = 77$ is selected.

$\beta \cdot \alpha^k = 13 \cdot 6^{77} \bmod 229 = 147 = 3 \cdot 7^2$

$\therefore \log_6 13 = (\log_6 3 + 2 \log_6 7 - 77) \bmod 228 = 117$.

Running time of Index Calculus Algorithm

- Computing discrete logarithms modulo a prime is only a little harder than factoring integers of the same size.
- With an optimal choice of t, the index-calculus algorithm for Z_p^* has an expected running time L_p [1/2, c] where $c > 0$ is a constant.

$$L_n\,[\alpha, c] = O(\exp((c\,(\ln n)^{\alpha}\,(\ln\ln n)^{1-\alpha}))$$

$$L_p\,[\,1/2, c] = O(\exp(c\,\sqrt{(\ln n \ln\ln n)}))$$

- The best algorithm known for computing logarithms in Z_p^* is index-calculus algorithm using number field sieve, with an expected running time of L_p [1/3, 1.923].

L-notation

- L-notation is an asymptotic notation
- It is analogous to big-O notation
- Denoted as $L_n[\alpha, c]$ for a bound variable $n \to \infty$.

  $$L_n[\alpha, c] = O(\exp((c (\ln n)^{\alpha} (\ln\ln n)^{1-\alpha}))$$

  where c is a positive constant, and α is a constant satisfying $0 < \alpha < 1$.
- L-notation is used mostly in computational number theory, to express the complexity of algorithms
- If $\alpha = 0$, then $L_n[0, c] = (\ln n)^c$ is a polynomial of $\ln n$.
- If $\alpha = 1$, then $L_n[1, c] = n^c$ is a polynomial in n.
- When $0 < \alpha < 1$, then the function is subexponential and the algorithm is subexponential time algorithm.