Algorithms for Computing Discrete Logarithms

Prof. Ashok K Bhateja

IIT Delhi

Discrete logarithm problem

- Definition DLP: Given a prime p, a generator α of Z_p^*, and an element $\beta \in Z_p^*$, find the integer x, $0 \leq x \leq p - 2$, s.t., $\alpha^x \equiv \beta \pmod{p}$.
- Let $p = 97$. Z_{97}^* is a cyclic group of order 96.

  A generator of Z_{97}^* is $\alpha = 5$.

  Since $5^{32} \equiv 35 \pmod{97}$ therefore $\log_5 35 = 32$ in Z_{97}^*.
- Definition GDLP: Given a finite cyclic group G of order n, a generator α of G, and an element $\beta \in G$, find the integer x, $0 \leq x \leq n - 1$, such that $\alpha^x \equiv \beta$.

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
- Number Field Sieve

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., $\beta \alpha^{-mq} = \alpha^r$
- 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 $\beta \alpha^{-mq} = \alpha^r$

    $x = mq + r$

Baby-step giant-step algorithm

Given: a generator α of a cyclic group $Z_p{}^*$ of order n and an element β in $Z_p{}^*$.

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

for $r = 0$ to m -1

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

Sort this table by second component (use hashing)

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 p

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

Sort by second row

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

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

$\therefore \log_3 57 \pmod{113} = 100$

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
- For all q where $0 \leq q < m$, it is required to search 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.

Pollard's rho algorithm for Discrete Logarithms

- Pollard an elegant algorithm proposed 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. $O(\sqrt{n})$
- 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, with generator α ; $\beta \in G$
- Find integers a, b, A, B s.t. $\alpha^{a} \beta^{b} = \alpha^{A} \beta^{B}$

  i.e., the equation $(B - b)\, x = (a - A)$ where $x = \log_{\alpha} \beta$
- For finding such a, b, A, B, Floyd's cycle-finding algorithm can be used which finds a cycle in the sequence $x_i = \alpha^{a_i}\beta^{b_i}$. i.e., find two group elements x_i and x_{2i} such that $x_i = x_{2i}$.
- Hence $\alpha^{a_i}\beta^{b_i} = \alpha^{a_{2i}}\beta^{b_{2i}} \Rightarrow \beta^{b_i - b_{2i}} = \alpha^{a_{2i} - a_i}$
- By taking log both sides $(b_i - b_{2i}) \log_{\alpha} \beta = (a_{2i} - a_i) \bmod n$ provided $b_i \neq b_{2i} \pmod{n}$. (note $b_i \equiv b_{2i} \bmod n$ occurs with probability ≈ 0)
- Solution to this equation can be easily obtained using Euclidean algorithm.

Floyd's cycle-finding

- Floyd's cycle-finding algorithm: For any function f that maps a finite set S to itself, and any initial value x_0 in S, the sequence of iterated function values

$$x_0,\ x_1 = f(x_0),\ x_2 = f(x_1),\ \ldots,\ x_i = f(x_{i-1}),\ \ldots$$

must eventually use the same value twice: there must be some pair of distinct indices i and j such that $x_i = x_j$.

- Cycle detection is the problem of finding i and j, given f and x_0.

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

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

- Choose a random integer $x_0 \in \{1, 2, \ldots, n\}$
  - The initial term $x_0 = \alpha^{a_0}\beta^{b_0}$ for random values of a_0 & b_0
  - a_0 & b_0 may be zero i.e. $x_0 = 1$
- Compute the sequence $\{x_i\}_{i \in N}$ using $x_{i+1} = f(x_i)$.

- $x_i = \alpha^{a_i}\beta^{b_i} \pmod{n}$ for $i \geq 0$
- The values of a_i and b_i can be obtained by

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

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

- Find two group elements x_i and x_{2i} using Floyd's cycle finding algorithm such that $x_i = x_{2i}$

$x_i = x_{2i}$ i.e. $\alpha^{a_i}\beta^{b_i} = \alpha^{a_{2i}}\,\beta^{b_{2i}} \pmod{n}$

$\therefore\ \beta^{b_i - b_{2i}} = \alpha^{a_{2i} - a_i} \pmod{n}$

$(b_i - b_{2i}) \log_\alpha \beta \equiv (a_{2i} - a_i) \bmod n$

$\therefore\ \log_\alpha \beta \equiv (b_i - b_{2i})^{-1}(a_{2i} - a_i) \bmod n$

provided $b_i \not\equiv b_{2i} \pmod{n}$

Floyd's cycle-finding algorithm

- One starts with the pair (x_1 , x_2) and iteratively computes (x_i , x_{2i}) from the previous (x_{i-1} , x_{2i-2}), until $x_m = x_{2m}$ for some m.
- The expected running time of this method is $O(n^{1/2})$.

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

Divide the set $S = \{1, 2, \ldots, 2016\}$ into 3 subsets S_0, S_1, S_2

$S_i = \{a \mid a \in S \;\&\; a = i \bmod 3\}$ 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, therefore

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

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

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

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

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

$$\equiv 1861 \times 1930 \bmod 2016$$

$$= 1234$$

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 n.

$$n = p_1^{e_1} \cdot p_2^{e_2} \cdots\ p_k^{e_k}$$

- Idea: compute $x \bmod p_i^{e_i}$ for each i, $1 \leq i \leq k$ then compute x mod n 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 $x = \log_\alpha \beta \pmod{n}$

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

Idea: compute $x \bmod p_i^{e_i}$ for each i, $1 \le i \le k$ then compute $x \pmod{n}$ 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

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

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

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

Step 1: to find x_0.

x_0 can be computed using the fact

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

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

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

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

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

$\equiv \alpha^{x_0\left(\frac{p-1}{q}\right)} \bmod p$ 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 $x_1, x_2, \ldots, x_{e\text{-}1}$ if $e > 1$

Define $\beta_0 = \beta$, and

$$\beta_j = \beta\alpha^{-(x_0+x_1q+\cdots+x_{j-1}q^{j-1})} \bmod p, \text{ for } 0 \le j \le e - 1$$

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)

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

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

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

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

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

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

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 $0 \le j \le e - 1$

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

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

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

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

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

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 $j \le e - 1$

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

$\quad$ find i such that $\delta = \gamma_i$

$\quad$ $x_j = i$

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

$\quad$ $j = j + 1$

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.

Pohlig-Hellman Algorithm: Analysis

- Given the factorization of n, the running time of the Pohlig-Hellman algorithm is $O\left(\sum_{i=1}^{n} e_i\left(\lg n + \sqrt{p_i}\right)\right)$group multiplications.
- Pohlig-Hellman algorithm is efficient only if each prime divisor p_i of n is relatively small; i.e., if n is a smooth integer.

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