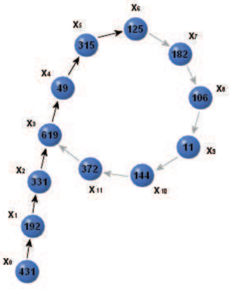
Discrete Logarithm Problem

Çetin Kaya Koç

http://cs.ucsb.edu/~koc

koc@cs.ucsb.edu



Exponentiation and Logarithms in $\mathcal{Z}_p^*$

- Consider the multiplicative group $\mathcal{Z}_p^*$ of integers modulo a prime p and a primitive element $g \in \mathcal{Z}_p^*$
- The exponentiation operation is the computation of y in

$$y = g^x = \overbrace{g \times g \times \cdots \times g}^{x \text{ times}} \pmod{p}$$

  for a positive integer x
- On the other hand, the discrete logarithm problem (DLP) is defined to be the computation of x, given y, g, and p
- This is the discrete analogue of the logarithm function

$$x = \log_g(y) \pmod{p}$$

Exponentiation and Logarithms in a General Group

- In a multiplicative group $(S, \otimes)$ with a primitive element $g \in S$, the exponentiation operation for a positive x is the computation of y in

$$y = g^x = \overbrace{g \otimes g \otimes \cdots \otimes g}^{x \text{ times}}$$

- On the other hand, in an additive group $(S, \oplus)$ with a primitive element $g \in S$, the point multiplication operation is the computation of y in

$$y = [x]g = \overbrace{g \oplus g \oplus \cdots \oplus g}^{x \text{ times}}$$

- In both cases, the discrete logarithm problem (DLP) is defined to be the computation of x, given y and g

Discrete Logarithms in Public-Key Cryptography

- If the DLP is difficult in a given group, we can use it to implement several public-key cryptographic algorithms, for example, Diffie-Hellman key exchange method, ElGamal public-key encryption method, and the Digital Signature Algorithm
- Two types of groups are noteworthy:
  - The multiplicative group $\mathcal{Z}_p^*$ of integers modulo a prime p
  - The additive group of elliptic curves defined over GF(p) or GF(2^k)
- The DLP problem in these groups are known to be difficult
- There may also be other groups worth considering, however, the DLP in additive mod p group is trivial, while the DLP in the multiplicative group of GF(2^k) is also shown to be rather easy (but not trivial)

Discrete Logarithms in $\mathcal{Z}_p^*$

- The **discrete logarithm problem** (DLP) is defined as the computation of $x \in \mathcal{Z}_p^*$ in

  $$y = g^x \pmod{p}$$

  given p, g, and y
- Example: Given $p = 23$ and $g = 5$, find x such that

  $$10 = 5^x \pmod{23}$$

  Answer: $x = 3$
- Example: Given $p = 23$ and $g = 5$, find x such that

  $$11 = 5^x \pmod{23}$$

  Answer: $x = 9$

Discrete Logarithms in $\mathcal{Z}_p^*$

- Given $p = 158(2^{800} + 25) + 1 =$

  1053546280395016975304616582933958731948871814925913489342
  6087342587178835751858673003862877377055779373829258737624
  5199045043066135085968269741025626827114728303489756321430
  0237166369174066615907176472549470083113107138189921280884
  003892629359

  and $g = 3$, find $x \in \mathcal{Z}_p^*$ such that

  $$2 = 3^x \pmod{p}$$

  Answer: ?

- How difficult is it to find x?

Exhaustive Search

- Since $x \in \mathcal{Z}_p^*$, we can perform search, and try all possible values of x:

  ```
  for i = 1 to p − 1
      if y = g^i (mod p) return x = i
  ```

- This would require $p - 1$ exponentiations
- If p requires k bits, a single exponentiation takes $O(k^3)$ arithmetic operations, and therefore, the number of arithmetic operations for performing the above search would be exponential in k

$$O(pk^3) = O(2^k k^3)$$

Shanks' Baby-Step-Giant-Step

- In 1973, Shanks described an algorithm for computing discrete logarithms that runs in $O(\sqrt{p})$ time and requires $O(\sqrt{p})$ space
- Let $y = g^x \pmod{p}$, with $m = \lceil \sqrt{p} \rceil$ and $p < 2^k$
- Shanks' method is a deterministic algorithm and requires the construction of two arrays S and T, which contains pairs of integers (u, v)
- The construction of S is called the giant-steps:

$$S = \{(i, g^{im}) \mid i = 0, 1, \ldots, m\}$$

- The construction of T is called the baby-steps:

$$T = \{(j, y \times g^j) \mid j = 0, 1, \ldots, m\}$$

Shanks' Baby-Step-Giant-Step

- To compute the discrete logarithm, find a group element that appears in both list, and get the indices i and j, and the solution x is then equal to

$$x = i \times m - j \pmod{n}$$

- To use this method in practice, one would typically only store the giant-steps array and the lookup each successive group element from the baby-steps array until a match is found
- However, the algorithm requires enormous amount of space, and thus, it is rarely used in practice
- Another method, called Pollard Rho method, has the same time complexity and requires negligible amount of space is preferred

Shanks' Baby-Step-Giant-Step

- Consider the solution of $y = 44 = 3^x \pmod{101}$
- $m = \lceil\sqrt{101}\rceil = 11$, therefore, the giant-steps and baby-steps tables:

$$S = \{(i, 3^{11i}) \mid i = 0, 1, \ldots, 11\}$$

i	0	1	2	3	4	5	6	7	8	9	10	11
3^{11i}	1	94	49	61	**78**	60	85	11	24	34	65	50

$$T = \{(j, 44 \times 3^j) \mid j = 0, 1, \ldots, 11\}$$

j	0	1	2	3	4	5	6	7	8	9	10	11
44×3^j	44	31	93	77	29	87	59	76	26	**78**	32	96

- The solution $x = 4 \times 11 - 9 = 35$, i.e., $3^{35} = 44 \pmod{101}$

Pollard Rho Algorithm for DLP

- Pollard Rho algorithm is also of $O(\sqrt{p})$ time complexity, however, it does not require a large table
- It forms a pseudorandom sequence of elements from the group, and searches for a cycle to appear in the sequence
- The sequence is defined deterministically and each successive element is a function of only the previous element
- If a group element appears a second time, every element of the sequence after that will be a repeat of elements in the sequence
- According to the birthday problem, a cycle should appear after $O(\sqrt{p})$ elements of the sequence have been computed

Pollard Rho Algorithm for DLP

- The standard version of the Pollard Rho algorithm defines the sequence

$$a_{i+1} = \begin{cases} y \times a_i & \text{for} \quad a_i \in S_1 \\ a_i^2 & \text{for} \quad a_i \in S_2 \\ g \times a_i & \text{for} \quad a_i \in S_3 \end{cases}$$

  where S_1, S_2, and S_3 are disjoint partitions of the group elements, that are approximately the same size
- The initial term is taken as $a_0 = g^{\alpha}$ for a random α
- There is no need to keep all of the group elements; we compute the sequences from a_i to a_{2i} until an equality is discovered
- The equality of two terms in the sequence implies equality on exponents modulo $(p-1)$ due to the Fermat's Theorem, from which we solve for x

- Consider the solution of $y = 44 = 3^x \pmod{101}$
- We divide the set $\{1, 2, \ldots, 100\}$ into 3 sets such that $S_1 = \{1, 2, \ldots, 33\}$, $S_2 = \{34, 35, \ldots, 66\}$, and $S_3 = \{67, 68, \ldots, 100\}$
- Starting with the first term $a_0 = g^\alpha$ for a random $\alpha = 15$, we get $a_0 = 3^{15} = 39$, and first few following terms of the iteration as

	a_i	S_1	S_2	S_3	a_i	$\log(y^u)$	$\log(g^v)$
$i = 0$	39		39		g^{15}	0	15
$i = 1$	$a_0^2 = 39^2$	6			g^{30}	0	30
$i = 2$	$y \cdot a_1 = 44 \cdot 6$		62		$y \cdot g^{30}$	1	30
$i = 3$	$a_2^2 = 62^2$	6			$y^2 \cdot g^{60}$	2	60
$i = 4$	$y \cdot a_3 = 44 \cdot 6$		62		$y^3 \cdot g^{60}$	3	60
$i = 5$	$a_4^2 = 62^2$	6			$y^6 \cdot g^{20}$	6	20

Pollard Rho Algorithm for DLP

- Therefore, we find $a_1 = a_3$ (also $a_2 = a_4$ and $a_3 = a_5$)
- The discovery of an equality in the sequence implies that we found a relationship between the exponent x and known powers of g
- The equality $a_1 = a_3$ implies

$$g^{30} = y^2 \cdot g^{60} = (g^x)^2 \cdot g^{60} = g^{2x+60}$$

Using Fermat's Little Theorem, we have an equality on the exponents

$$30 = 2x + 60 \pmod{100} \rightarrow 2x = 70 \pmod{100}$$

Since $\gcd(2, 100) \neq 1$, this equation has two solutions: $x = \{35, 85\}$

Pollard Rho Algorithm for DLP

- We can take each solution and check whether they verify:

$$3^x \stackrel{?}{=} 44 \pmod{100}$$

We see that $x = 35$ is a solution since $3^{35} = 44 \pmod{101}$

- Similarly, the equality of $a_2 = a_4$ gives the same equation:

$$x + 30 = 3x + 60 \pmod{100} \rightarrow 2x = 70 \pmod{100}$$

- On the other hand, the equality of $a_3 = a_5$ implies

$$2x + 60 = 6x + 20 \pmod{100} \rightarrow 4x = 40 \pmod{100}$$

We find 4 possible solutions $x = \{10, 35, 60, 85\}$

Discrete Logarithm Problem

- The Pollard Rho algorithm is generating a sequence and hoping to find a match, due to the birthday problem
- Its time complexity is $O(\sqrt{p})$ which is still exponential in terms of the input size in bits: $O(2^{k/2})$
- However, there are subexponential algorithms, for example the index calculus method for the group $\mathcal{Z}_p^*$ has subexponential time complexity
- On the hand, the discrete logarithm problem for the elliptic curve groups remains to be a formidable problem
- There is no subexponential algorithm for the elliptic curve discrete logarithm problem (ECDLP) as of yet