Introduction to Cryptology

13.2 - Generic Discrete-Logarithm Algorithms

Federico Pintore
Mathematical Institute, University of Oxford (UK)

UNIVERSITY OF OXFORD

Michaelmas term 2020

Why Discrete Logarithm?

Consider the prime $p = 941$ and the group $\mathbb{Z}_p^*$.

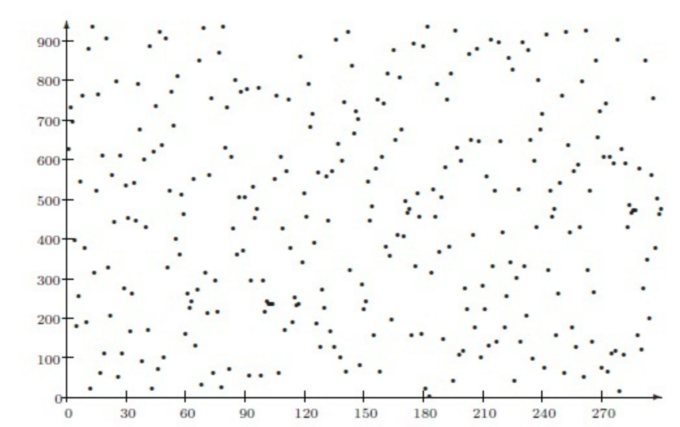

Figure Graph of $f(x) = 627^x \pmod{941}$ for $x = 1, 2, 3, \ldots$

Discrete logarithms

Computing discrete logarithms in $\mathbb{G} = (\mathbb{Z}_q, +)$ is easy.

Recent advancements for $\mathbb{G} = (\mathbb{F}^*_{2^n}, \cdot)$ (more generally, for fields of small characteristic).

Computing discrete logarithms in $\mathbb{G} = \mathbb{Z}^*_p$ is believed to be hard, and even harder in (well-chosen) groups of elliptic curves.

Generic algorithms

Generic algorithms do not exploit any special properties of the group elements, and apply to arbitrary groups.

They include

- exhaustive search,
- BSGS,
- Pollard's Rho.

There exist better algorithms for multiplicative groups of finite fields. Still no better algorithms for (well-chosen) elliptic curves.

Exhaustive search

Input: group $\mathbb{G}$ and $g, h \in \mathbb{G}$ s.t. $g^x = h$

Output: x

$k \leftarrow 1$

$h' \leftarrow g$

if $h' = h \quad (\star)$

  return k

else

  $k \leftarrow k + 1;$

  $h' \leftarrow h'g$

  go to $(\star)$

Exhaustive search

Input: group $\mathbb{G}$ and $g, h \in \mathbb{G}$ s.t. $g^x = h$

Output: x

$k \leftarrow 1$

$h' \leftarrow g$

if $h' = h \quad (\star)$

  return k

else

  $k \leftarrow k + 1;$

  $h' \leftarrow h'g$

  go to $(\star)$

The worst-case complexity is $|\mathbb{G}|$.

Pohlig-Hellman Algorithm

It shows that the Dlog problem in a cyclic group $\mathbb{G}$ is as hard as the Dlog problem in the largest subgroup of prime order in $\mathbb{G}$.

Assume $|\mathbb{G}| = N = pq$, and let g be a generator of $\mathbb{G}$.

Observe that g^p generates a subgroup of order q, and $h = g^x$ implies $h^p = (g^p)^x$.

Solving the Dlog problem with input $(\langle g^p \rangle, h^p, g^p)$ determines $x \pmod{q}$. Analogously, it is possible to determine $x \pmod{p}$.

Chinese Reminder Theorem: given $a \in \{0, \ldots, pq-1\}$, $[a]_{pq}$ is uniquely determined by $[a]_p$ and $[a]_q$.

Pohlig-Hellman Algorithm

Assume $|\mathbb{G}| = p^e$, and let g be a generator of $\mathbb{G}$.

The discrete logarithm x can be written as

$$x_0 + x_1 p + \cdots + x_{e-1} p^{e-1}$$

with $0 \leq x_i < p$.

Observe that $g^{p^{e-1}}$ generates a subgroup of order p, and $h = g^x$ implies $h^{p^{e-1}} = (g^{p^{e-1}})^x$.

Solving the Dlog problem with input $(\langle g^{p^{e-1}} \rangle, h^{p^{e-1}}, g^{p^{e-1}})$ determines $x \pmod{p}$, i.e. x_0.

Consider $h_1 = h \cdot g^{-x_0}$. Then $h_1 = g^{x_0 + x_1 p + \cdots + x_{e-1} p^{e-1} - x_0} =$ $= (g^p)^{x_1 + \cdots + x_{e-1} p^{e-2}}$, and x_1 can be obtained form h_1 and g^p.

Pohlig-Hellman Algorithm

More in general, suppose $\mathbb{G} = \langle g \rangle$ is of order $N = \prod_{i=1}^{\ell} p_i^{e_i}$.

Observe that $g^{N/p_i^{e_i}}$ generates a subgroup of order $p_i^{e_i}$, and $h = g^x$ implies $h^{N/p_i^{e_i}} = (g^{N/p_i^{e_i}})^x$.

Solving the Dlog problem with input $(\langle g^{N/p_i^{e_i}} \rangle, h^{N/p_i^{e_i}}, g^{N/p_i^{e_i}})$ determines $x \pmod{p_i^{e_i}}$.

Chinese Reminder Theorem: given $a \in \{0, \ldots, N-1\}$, $[a]_N$ is uniquely determined by the congruence classes $[a]_{p_1^{e_1}}, \ldots, [a]_{p_\ell^{e_\ell}}$.

Baby-Step/Giant-Step (BSGS)

Thanks to the Pohlig-Hellman algorithm, we can restrict ourselves to cyclic groups $\mathbb{G} = \langle g \rangle$ of prime order p.

Baby-Step/Giant-Step (BSGS)

Thanks to the Pohlig-Hellman algorithm, we can restrict ourselves to cyclic groups $\mathbb{G} = \langle g \rangle$ of prime order p.

The Baby-Step/Giant-Step algorithm works as follows:

- Let $N' = \lceil \sqrt{|\mathbb{G}|} \rceil$.
- There exist $0 \leq i, j < N'$ such that $x = jN' + i$. Therefore:

$$h = g^{jN'+i} \Leftrightarrow hg^{-jN'} = g^i.$$

- Compute $L_B := \{g^i | i = 0, \ldots, N' - 1\}$.
- Compute $L_G := \{hg^{-jN'} | j = 0, \ldots, N' - 1\}$.

Baby-Step/Giant-Step (BSGS)

Thanks to the Pohlig-Hellman algorithm, we can restrict ourselves to cyclic groups $\mathbb{G} = \langle g \rangle$ of prime order p.

The Baby-Step/Giant-Step algorithm works as follows:

- Let $N' = \lceil \sqrt{|\mathbb{G}|} \rceil$.
- There exist $0 \leq i, j < N'$ such that $x = jN' + i$. Therefore:

$$h = g^{jN'+i} \Leftrightarrow hg^{-jN'} = g^i.$$

- Compute $L_B := \{g^i | i = 0, \ldots, N' - 1\}$.
- Compute $L_G := \{hg^{-jN'} | j = 0, \ldots, N' - 1\}$.

The algorithm requires time and memory $\mathcal{O}\left(|\mathbb{G}|^{1/2}\right)$.

Pollard's Algorithms

John Pollard is a famous name in the field of factoring/Dlog algorithms.

He is known for:

- the $(p - 1)$ method,
- the Rho algorithm,
- the Number Field Sieve.

Pollard's Rho Algorithm

The idea used in the Rho algorithm is to find a collision for a random map f.

Similarly to the better birthday attack for hash functions, the Floyd's cycle finding algorithm is used, i.e. given (x_i, x_{2i}),

$$(x_{i+1}, x_{2i+2}) = (f(x_i), f(f(x_{2i})))$$

are computed.

The algorithm stops when $x_\ell = x_{2\ell}$.

Pollard's Rho Algorithm

Define the subsets G_1, G_2, G_3 of about the same size and such that $\mathbb{G} = G_1 \cup G_2 \cup G_3$ and $G_i \cap G_j = \emptyset$.

On input $g, h = g^x$, define a random map $f : G \to G$ such that

$$x_{i+1} = f(x_i) := \begin{cases} hx_i & x_i \in G_1 \\ x_i^2 & x_i \in G_2 \\ gx_i & x_i \in G_3 \end{cases}$$

Pollard's Rho Algorithm

- Set x_0 to 1 and apply f recursively to get $\{x_i, x_{2i}\}_i$
- At each iteration, the algorithm stores (x_i, a_i, b_i) and $(x_{2i-2}, a_{2i-2}, b_{2i-2})$, where (x_i, a_i, b_i) is denoted by $f(x_{i-1}, a_{i-1}, b_{i-1})$, s.t. $x_i = g^{a_i} h^{b_i}$, and:

$$(a_i, b_i) = \begin{cases} (a_{i-1}, b_{i-1} + 1 \pmod p) & x_{i-1} \in G_1 \\ (2a_{i-1} \pmod p, 2b_{i-1} \pmod p) & x_{i-1} \in G_2 \\ (a_{i-1} + 1 \pmod p, b_{i-1}) & x_{i-1} \in G_3. \end{cases}$$

- The algorithm stops when a collision is found, i.e. $x_\ell = x_{2\ell}$. Therefore

$$x = \frac{a_{2\ell} - a_\ell}{b_\ell - b_{2\ell}} \pmod p.$$

If f is "random enough", a collision is expected to be found in time $\mathcal{O}\left(\sqrt{|G|}\right)$, while only two triples are stored at each step.

Pollard's Rho Algorithm

Input: group $\mathbb{G}$ and $g, h \in \mathbb{G}$ s.t. $g^x = h$

Output: x

$N \leftarrow \lceil \sqrt{|\mathbb{G}|} \rceil$

$a_1 = 0;\ b_1 = 0;\ x_1 = 1$

$(x_2, a_2, b_2) = f(x_1, a_1, b_1)$

for $k \in \{2, \ldots, N\}$

  $(x_1, a_1, b_1) = f(x_1, a_1, b_1)$

  $(x_2, a_2, b_2) = f(f(x_2, a_2, b_2))$

  if $x_1 = x_2$

    break

if $b_1 = b_2 \pmod p$

  return $\perp$

else

  return $(a_2 - a_1)/(b_1 - b_2) \pmod p$

Pollard's Rho Algorithm: Example

Example (Smart's book)

Consider $\mathbb{G} = \langle g \rangle$, with $g = 64 \in \mathbb{Z}_{607}^*$. $\mathbb{G}$ has order $p = 101$.

Given $h = 122 = 64^x$, the problem is to determine x.

$\langle g \rangle$ can be splitted into three sets G_1, G_2, G_3 as follows:

$$G_1 = \{x \in \mathbb{F}_{607}^* : 0 \leq x \leq 201\}$$

$$G_2 = \{x \in \mathbb{F}_{607}^* : 202 \leq x \leq 403\}$$

$$G_3 = \{x \in \mathbb{F}_{607}^* : 404 \leq x \leq 606\}$$

Pollard's Rho: example

Example (Smart's book)

i	x_i	a_i	b_i	x_{2i}	a_{2i}	b_{2i}
0	1	0	0	1	0	0
1	122	0	1	316	0	2
2	316	0	2	172	0	8
3	308	0	4	137	0	18
4	172	0	8	7	0	38
5	346	0	9	309	0	78
6	137	0	18	352	0	56
7	325	0	19	167	0	12
8	7	0	38	498	0	26
9	247	0	39	172	2	52
10	309	0	78	137	4	5
11	182	0	55	7	8	12
12	352	0	56	309	16	26
13	76	0	11	352	32	53
14	167	0	12	167	64	6

A collision is found when $i = 14$, which implies $g^0h^{12} = g^{64}h^6$, so $12x = 64 + 6x \pmod{101}$, and therefore $x = 78$.

More from Pollard

Pollard's Lambda Method: it is similar to the Rho Algorithm (it uses deterministic random walk), but it is tailored to the cases where it is known that the Dlog lies in a particular interval.

Parallel Pollard's Rho Algorithm: it is designed to use computing resources of different sites across the internet.

Further Reading I

Andrew Granville.
Smooth numbers: computational number theory and beyond.
Algorithmic number theory: lattices, number fields, curves and cryptography, 44:267–323, 2008.

Antoine Joux, Andrew Odlyzko, and Cécile Pierrot.
The past, evolving present, and future of the discrete logarithm.
In Open Problems in Mathematics and Computational Science, pages 5–36. Springer, 2014.

Carl Pomerance.
Smooth numbers and the quadratic sieve.
Algorithmic Number Theory, Cambridge, MSRI publication, 44:69–82, 2008.

Further Reading II

Carl Pomerance.
A tale of two sieves.
Biscuits of Number Theory, 85, 2008.

Victor Shoup.
Lower bounds for discrete logarithms and related problems.
In Advances in Cryptology—EUROCRYPT'97, pages 256–266. Springer, 1997.