Discrete Logarithm Algorithms

Ali El Kaafarani

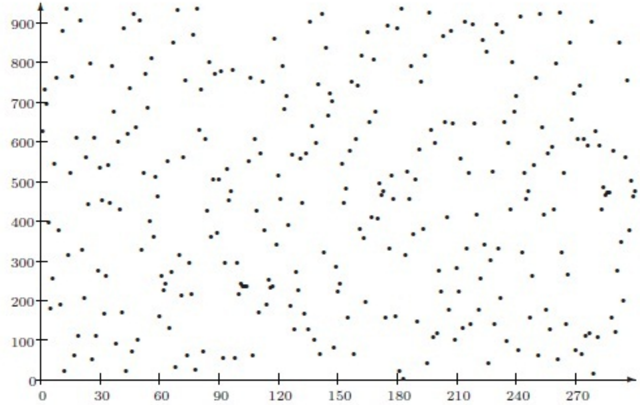
[1]Mathematical Institute
[2] PQShield Ltd.

Outline

1. Generic discrete logarithm algorithms
2. Discrete logarithms over finite fields

Why Discrete Logarithm?

A graph of $f(x) = 627^x \bmod 941$ for $x = 1, 2, 3, \ldots$



Discrete logarithms

- Trivial if $(G, \circ) = (\mathbb{F}_p, +)$. Why?
- Recently broken if $(G, \circ) = (\mathbb{F}^*_{2^n}, *)$
  (more generally if characteristic is small)
- Believed to be hard for $G = \mathbb{F}^*_p$
  and harder for (well-chosen) elliptic curve groups

Discrete logarithms

- Trivial if $(G, \circ) = (\mathbb{F}_p, +)$. Why?
- Recently broken if $(G, \circ) = (\mathbb{F}_{2^n}^*, *)$
  (more generally if characteristic is small)
- Believed to be hard for $G = \mathbb{F}_p^*$
  and harder for (well-chosen) elliptic curve groups

Discrete logarithms

- Trivial if $(G, \circ) = (\mathbb{F}_p, +)$. Why?
- Recently broken if $(G, \circ) = (\mathbb{F}_{2^n}^*, *)$
  (more generally if characteristic is small)
- Believed to be hard for $G = \mathbb{F}_p^*$
  and harder for (well-chosen) elliptic curve groups

Outline

1. Generic discrete logarithm algorithms
2. Discrete logarithms over finite fields

Outline

Generic group model

- Algorithms do not exploit any special properties of the encodings of the group elements, other than the fact that each group element is encoded as a unique binary string.
- For instance, the attacker just receives bitstrings instead of $\mathbb{Z}_n$ elements (n itself is often hidden but the size of n cannot be hidden).
- Operations on group elements are performed using an oracle that provides access to the group operations.
- Some attacks are generic: they work for any group.
- This includes exhaustive search, BSGS, Pollard's rho
- There exist much better attacks for finite fields.
- Still no better attack for (well-chosen) elliptic curves.

Exhaustive search

- Given $g, h \in G$ do the following

  1: $k \leftarrow 1; h' \leftarrow g$
  2: **if** $h' = h$ **then**
  3:     **return** k
  4: **else**
  5:     $k \leftarrow k + 1; h' \leftarrow h'g$
  6:     Go to Step 2
  7: **end if**

- Generic algorithm
- Time complexity $|G|$ in the worst case, $|G|/2$ on average
- Can we do better?

Pohlig-Hellman

- They observed that Dlog in a group $\mathbb{G}$ is as hard as the Dlog in the largest subgroup of prime order in $\mathbb{G}$.
- This applies in any arbitrary finite abelian group.
- Assume $|\mathbb{G}| = N = n_1 n_2$ and let g a generator of G.
- $h = g^k$ implies $h^{n_1} = (g^{n_1})^k$
  where g^{n_1} generates a subgroup of order n_2.
- Assuming that we can solve DLP in that subgroup, this would give us $k \bmod n_2$.
- Repeating the same thing for each factor of N and using CRT would give us k.

Pohlig-Hellman

- Let $\mathbb{G} = \langle g \rangle$ of order $N = \#\mathbb{G} = \prod_{i=1}^{\ell} p_i^{e_i}$
- Given $h = g^x$, we want to first find $x \mod p_i^{e_i}$ and then use CRT to recover it mod N.
- There is a group isomorphism $\phi : \mathbb{G} \to C_{p_1^{e_1}} \times \cdots \times C_{p_\ell^{e_\ell}}$.
- Define the projection map $\phi_{p_i} : \mathbb{G} \to C_{p_i^{e_i}}$ where $\phi_{p_i}(g) = g^{N/p_i^{e_i}}$. ϕ_{p_i} is a group homomorphism, i.e., if $h = g^x$ in $\mathbb{G}$, then $\phi_{p_i}(h) = \phi_{p_i}(g)^x$ in $C_{p_i^{e_i}}$.
- Solving the discrete logarithm in $C_{p_i^{e_i}}$ reduces to solving e_i discrete logarithm in the group C_{p_i} following an inductive procedure.
- Given $h' = g^{x'} \in C_{p_i^{e_i}}$, we write $x' = x_0 + x_1 p_i + \cdots + x_{e_i - 1} p_i^{e_i - 1}$ and then find $x_0, x_1, \ldots, x_{e_i-1}$ in turn.

Baby step, giant step (BSGS)

- Given a public cyclic group $\mathbb{G} = \langle g \rangle$, now we can assume that $\mathbb{G}$ has a prime order p.
- Given $h \in \mathbb{G}$, find the value of k s.t. $h = g^k$.
- Let $N' = \lceil \sqrt{|\mathbb{G}|} \rceil$
- There exist $0 \leq i, j < N'$ such that $k = jN' + i$

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

- Compute $L_B := \{g^i | i = 0, \ldots, N' - 1\}$
- Compute $L_G := \{hg^{-jN'} | j = 0, \ldots, N' - 1\}$
- Attack requires time and memory each $\mathcal{O}\left(|\mathbb{G}|^{1/2}\right)$
- Can we do better in terms of space requirement and still obtain a time complexity of $\mathcal{O}\left(\sqrt{|\mathbb{G}|}\right)$

Baby step, giant step (BSGS)

- Given a public cyclic group $\mathbb{G} = \langle g \rangle$, now we can assume that $\mathbb{G}$ has a prime order p.
- Given $h \in \mathbb{G}$, find the value of k s.t. $h = g^k$.
- Let $N' = \lceil \sqrt{|\mathbb{G}|} \rceil$
- There exist $0 \leq i, j < N'$ such that $k = jN' + i$

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

- Compute $L_B := \{g^i | i = 0, \ldots, N' - 1\}$
- Compute $L_G := \{hg^{-jN'} | j = 0, \ldots, N' - 1\}$
- Attack requires time and memory each $\mathcal{O}\left(|\mathbb{G}|^{1/2}\right)$
- Can we do better in terms of space requirement and still obtain a time complexity of $\mathcal{O}\left(\sqrt{|\mathbb{G}|}\right)$

Baby step, giant step (BSGS)

- Given a public cyclic group $\mathbb{G} = \langle g \rangle$, now we can assume that $\mathbb{G}$ has a prime order p.
- Given $h \in \mathbb{G}$, find the value of k s.t. $h = g^k$.
- Let $N' = \lceil \sqrt{|\mathbb{G}|} \rceil$
- There exist $0 \leq i, j < N'$ such that $k = jN' + i$

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

- Compute $L_B := \{g^i | i = 0, \ldots, N' - 1\}$
- Compute $L_G := \{hg^{-jN'} | j = 0, \ldots, N' - 1\}$
- Attack requires time and memory each $\mathcal{O}\left(|\mathbb{G}|^{1/2}\right)$
- Can we do better in terms of space requirement and still obtain a time complexity of $\mathcal{O}\left(\sqrt{|\mathbb{G}|}\right)$

Baby step, giant step (BSGS)

- Given a public cyclic group $\mathbb{G} = \langle g \rangle$, now we can assume that $\mathbb{G}$ has a prime order p.
- Given $h \in \mathbb{G}$, find the value of k s.t. $h = g^k$.
- Let $N' = \lceil \sqrt{|\mathbb{G}|} \rceil$
- There exist $0 \leq i, j < N'$ such that $k = jN' + i$

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

- Compute $L_B := \{g^i | i = 0, \ldots, N' - 1\}$
- Compute $L_G := \{hg^{-jN'} | j = 0, \ldots, N' - 1\}$
- Attack requires time and memory each $\mathcal{O}\left(|\mathbb{G}|^{1/2}\right)$
- Can we do better in terms of space requirement and still obtain a time complexity of $\mathcal{O}\left(\sqrt{|\mathbb{G}|}\right)$

Baby step, giant step (BSGS)

- Given a public cyclic group $\mathbb{G} = \langle g \rangle$, now we can assume that $\mathbb{G}$ has a prime order p.
- Given $h \in \mathbb{G}$, find the value of k s.t. $h = g^k$.
- Let $N' = \lceil \sqrt{|\mathbb{G}|} \rceil$
- There exist $0 \leq i, j < N'$ such that $k = jN' + i$

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

- Compute $L_B := \{g^i | i = 0, \ldots, N' - 1\}$
- Compute $L_G := \{hg^{-jN'} | j = 0, \ldots, N' - 1\}$
- Attack requires time and memory each $\mathcal{O}\left(|\mathbb{G}|^{1/2}\right)$
- Can we do better in terms of space requirement and still obtain a time complexity of $\mathcal{O}\left(\sqrt{|\mathbb{G}|}\right)$

Baby step, giant step (BSGS)

- Given a public cyclic group $\mathbb{G} = \langle g \rangle$, now we can assume that $\mathbb{G}$ has a prime order p.
- Given $h \in \mathbb{G}$, find the value of k s.t. $h = g^k$.
- Let $N' = \lceil \sqrt{|\mathbb{G}|} \rceil$
- There exist $0 \leq i, j < N'$ such that $k = jN' + i$

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

- Compute $L_B := \{g^i | i = 0, \ldots, N' - 1\}$
- Compute $L_G := \{hg^{-jN'} | j = 0, \ldots, N' - 1\}$
- Attack requires time and memory each $\mathcal{O}\left(|\mathbb{G}|^{1/2}\right)$
- Can we do better in terms of space requirement and still obtain a time complexity of $\mathcal{O}\left(\sqrt{|\mathbb{G}|}\right)$

Pollard's Algorithms

- John Pollard, a famous name in factoring/Dlog algorithms in the 20th century.
- Known for $(P-1)$ method, Rho-method, Number Field Sieve.
- The idea in the Rho method is to find a collision in a random mapping.
- Using the birthday paradox *naively* is no better than Baby-Step/Giant-Step method in terms of space/time requirements.
- Similar to the improved birthday paradox attack on hash functions, we can use Floyd's cycle finding algorithm, i.e. given (x_i, x_{2i}), we compute

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

- We stop when $x_\ell = x_{2\ell}$

Pollard's rho

- Define the sets G_1, G_2, G_3 of about the same size such that $G = G_1 \cup G_2 \cup G_3$ and $G_i \cap G_j = \{\}$, assuming that $1 \notin G_2$.
- Over $\mathbb{Z}_p^*$, one can choose
  $G_1 = \{0, \ldots, \lfloor p/3 \rfloor\}$,
  $G_2 = \{\lfloor p/3 \rfloor + 1, \ldots, \lfloor 2p/3 \rfloor\}$, $G_3 = \{\lfloor 2p/3 \rfloor + 1, \ldots, p-2\}$
- Define a random walk $f : G \to G$ 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

- Given $g, h = g^x$, we start from $x_0 := 1$ and apply f recursively to get $\{x_i, x_{2i}\}_i$.
- By the way f is defined, we can keep track of (x_t, a_t, b_t) such that $x_t = g^{a_t} h^{b_t}$, where

$$a_{i+1} = \begin{cases} a_i \\ 2a_i \mod p \\ a_i + 1 \mod p \end{cases}, b_{i+1} = \begin{cases} b_i + 1 \mod p & x_i \in G_1 \\ 2b_i \mod p & x_i \in G_2 \\ b_i & x_i \in G_3 \end{cases}$$

- We stop when a collision is found, i.e. $x_\ell = x_{2\ell}$, therefore $x = \dfrac{a_{2\ell} - a_\ell}{b_\ell - b_{2\ell}} \mod p$.
- If f is "random enough", then we should find the Dlog in expected time $\mathcal{O}\left(\sqrt{|G|}\right)$.

Pollard's rho

$N \leftarrow \lceil \sqrt{|G|} \rceil$
$a_1 = 0; b_1 = 0; x_1 = 1$
$(x_2, a_2, b_2) = f(x_1, a_1, b_1)$
for $k \in \{2, \ldots, N\}$ **do**
    $(x_1, a_1, b_1) = f(x_1, a_1, b_1)$
    $(x_2, a_2, b_2) = f(f(x_2, a_2, b_2))$
    if $x_1 = x_2$ **break;**
end for
if $b_1 = b_2 \mod p$ **then**
    return $\perp$
else
    return$(a_2 - a_1)/(b_1 - b_2) \mod p$
end if

Pollard's Rho: example

Example (Smart's book)

Consider $\mathbb{G} = \langle g \rangle$, a subgroup of $\mathbb{F}_{607}^*$ of order $p = 101$, with $g = 64$. Given $h = 122 = 64^x$. Solve for x.
We split $\mathbb{G}$ into three sets S_1, S_2, S_3 as follows:

$$S_1 = \{x \in \mathbb{F}_{607}^* : x \leq 201\}$$

$$S_2 = \{x \in \mathbb{F}_{607}^* : 202 \leq x \leq 403\}$$

$$S_3 = \{x \in \mathbb{F}_{607}^* : 404 \leq x \leq 606\}$$

Pollard's Rho: example

Example

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 $i = 14$, this implies that $g^0 h^{12} = g^{64} h^6$, so $[12x = 64 + 6x \mod 101]$ and therefore $x = 78$.

More from Pollard

- Pollard's Lambda Method: similar to the Rho method in that it uses deterministic random walk, but it is particularly designed to the cases where we know that the Dlog lies in a particular interval.
- Parallel Pollard's Rho: designed to be able to use computing resources of different sites across the internet.

Outline

1. Generic discrete logarithm algorithms

2. Discrete logarithms over finite fields

L notation

$$L_Q(\alpha; c) = \exp(c(\log Q)^\alpha(\log\log Q)^{1-\alpha})$$

- Q is the size of the field
- $\alpha = 0 \Rightarrow L_Q(\alpha; c) = (\log Q)^c$ polynomial
- $\alpha = 1 \Rightarrow L_Q(\alpha; c) = Q^c$ exponential

(simplified) Index Calculus for $\mathbb{F}_p^*$

- DLP: given $g, h \in \mathbb{F}_p^*$, find x such that $h = g^x$
- Factor basis made of **small primes**

$$\mathcal{F}_B := \{\text{primes } p_i \leq B\} = \{p_1, \ldots, p_k\}$$

- **Relation search**
  - Compute $g_i := g^{a_i}$ for random $a_i \in \{1, \ldots, p-1\}$
  - **If** all factors of g_i are $\leq B$, we have a relation

$$g^{a_i} = \prod_{p_j \in \mathcal{F}} p_j^{e_{i,j}} \tag{1}$$

- **Linear algebra** Once we have $\ell \geq k$ linearly independent equations similar to equations (1), we solve $\mod (p-1)$ for $\log_g p_i$, $i = 1, \ldots, k$.
- Search for t such that $[g^t \cdot h \mod p]$ is B-smooth. Once found, solve for $\log_g h \mod (p-1)$.

Size of the factor basis

- By the prime number theorem,

$$|\{\text{primes } p_i \leq B\}| \approx \frac{B}{\log B}$$

- Fact: 30% of all numbers have no prime factors above their square root. Surprisingly, a large proportion of numbers can be built out of so few primes!

Complexity Analysis

- How to choose an optimal B: If B is large, then it is more likely that the generated elements are B-smooth, but then testing that they are B-smooth is more difficult now. Therefore, we need to balance the cost!
- In order to choose an optimal B, we also need to know the probability that a random integer that is smaller than N is B-smooth.
- We will assume that the cost of generating relations dominates the overall complexity of Algorithm, i.e. assume that the linear algebra is negligible in terms of time complexity.
- We will simply use the trial-division to factor over $\mathcal{F}_B$.

Complexity Analysis

- A number is B-smooth if all its prime factors are smaller than B.
- Define $\Psi(N, B) = \#\{B\text{-smooth numbers} \leq N\}$.
- The probability that a positive integer $m \leq N$ is B-smooth is approximately equal to $\frac{1}{N} \cdot \Psi(N, B)$.
- The Canfield-Erdos-Pomerance Theorem: Let $u = \frac{\log N}{\log B}$, we have $\frac{1}{N} \cdot \Psi(N, B) = u^{-u+o(u)}$. This is the *Dickman-de Bruijn* function ρ, i.e. $\rho(u) \approx u^{-u}$.
- The expected number of random trials of choosing numbers in $[1; N]$ to find one that is B-smooth is $\approx u^u$

Complexity Analysis

- Let $|\mathcal{F}_B| = k$, the expected running time of the algorithm is

$$\approx \underbrace{(k+1)}_{\text{nb of relations}} \cdot \underbrace{u^u}_{\text{expected nb of trials}} \cdot \underbrace{k}_{\text{nb of trial divisions}} \cdot \underbrace{M(\log N)}_{\text{time for a trial division}} \tag{2}$$

$$\approx B^2 \cdot u^u \quad \textit{drop the logarithmic factors, where } k \approx \frac{B}{\log B} \tag{3}$$

$$= N^{2/u} \cdot u^u \tag{4}$$

- We want to minimize $f(u) = N^{2/u} \cdot u^u$. If we set $f'(u) = 0$, we need a u s.t. $u^2 \log u \approx 2 \log N$.
- Let $u = 2\sqrt{\frac{\log N}{\log \log N}}$, we then get $u^2 \log u = 2 \log N + o(\log N)$

Complexity Analysis

- Back to our bound B:

$$\begin{aligned} B &= N^{1/u} \\ &= exp(\frac{1}{u}\log N) \\ &= exp(\frac{1}{2}\sqrt{\log N \log\log N}) \\ &= L_N(1/2, 1/2) \end{aligned}$$

- Note that $u^u = L_N(1/2, 1)$, therefore $B^2 u^u = L_N(1/2, 2)$.
- The cost of the linear algebra step is bounded by $\tilde{O}(B^3)$, i.e. $L_N(1/2, 3/2)$.

Further Reading (1)

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 (2)

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.