The Discrete Logarithm Problem

Christophe Clavier

University of Limoges

Master 2 Cryptis

Problem Statement

The Discrete Logarithm Problem

Given $\mathbb{G}$ a (multiplicatively written) finite cyclic group of order n, given α a generator of $\mathbb{G}$ and $\beta \in \mathbb{G}$, find $0 \leq x \leq n-1$ such that $\beta = \alpha^x$.

Notation: $x = \log_\alpha \beta$

This statement of the discrete log problem is generic. No particular assumption is formulated about $\mathbb{G}$.

Example

Consider $p = 97$ prime.
$\mathbb{Z}_{97}^\star$ is a cyclic group of order 96. One of its generators is $\alpha = 5$.
As $5^{32} \equiv 35 \pmod{97}$, we have $\log_5 35 = 32$ in $\mathbb{Z}_{97}^\star$.

The Diffie Hellman Key Agreement Protocol

The Diffie-Hellman key exchange was the first system of public key type. It was invented at the same time than public key cryptography, in 1976. It is used to exchange a key between two users, say Alice and Bob, that do not know each other.

Let's say that all users use a common group $\mathbb{G} = \langle g \rangle$ of order q.
($\mathbb{G}$, g and q are system (public) parameters.)

To share a key, Alice generates a random $x_a \in \mathbb{Z}_q^\star$ and Bob generates his $x_b \in \mathbb{Z}_q^\star$. Alice sends $y_a = g^{x_a}$ to Bob, which replies with $y_b = g^{x_b}$.

Now, Alice computes $K = {y_b}^{x_a}$, and on his side, Bob can recover K by formula $K = {y_a}^{x_b}$. Now, Alice and Bob have a shared secret to communicate securely over an open channel (via secret-key primitives, for example).

Recovering K from y_a and y_b is as difficult as solving the discrete logarithm problem.

Other schemes

Many other cryptographic schemes or protocols are based on the difficulty of computing discrete logarithms in large cyclic groups:

- The Schnorr identification protocol
- The Schnorr signature scheme
- The El Gamal encryption scheme
- The Digital Signature Algorithm (DSA) (a NIST standard)
- Elliptic Curve based Diffie Hellman key exchange (ECDH)
- Elliptic Curve based Digital Signature Algorithm (ECDSA)
- ...

Classification of Solving Methods

Methods for solving the discrete logarithm problem can be split in the following categories:

- Generic methods which apply in any subgroup: exhaustive search, baby-step giant-step, Pollard's rho
- Methods which apply in any subgroup but are particularly efficient when the group order is smooth: Pohlig-Hellmann
- Dedicated methods which are efficient only in particular groups: index calculus (in $\mathbb{Z}_p^\star$ for instance)

Exhaustive Search

This method simply consists in evaluating successive powers of α until one of them is equal to β.

It is straightforward but particularly inefficient except when the group is of small order. Typically, only use this method for $n \lesssim 100$.

Baby-step Giant-step

Principle

- Time-memory tradeoff of the exhaustive search based on the following observation:
  - For $m = \lceil \sqrt{n} \rceil$, one can write $x = \log_\alpha \beta$ as $x = im + j$ with $0 \leq i, j < m$.
  - For those i, j the following holds: $\beta(\alpha^{-m})^i = \alpha^j$
- One can build a table of all α^j for $0 \leq j < m$, and search for some $\beta(\alpha^{-m})^i$ present in the table.

  A collision for the couple (i, j) reveals $x = im + j$.

Baby-step Giant-step

Algorithm

Algorithm 1 Baby-step giant-step algorithm

Input: A group $\mathbb{G}$ of order n, $\alpha \in \mathbb{G}$ a generator, $\beta \in \mathbb{G}$
Output: $x = \log_\alpha \beta$

```
m ← ⌈√n⌉
For all 0 ≤ j < m, compute α^j and store (α^j, j) in a table
Compute α^(−m),   γ ← β
for i = 0 to m − 1 do
    if (γ, j) appears in the table for some j then
        return  x = im + j
    γ ← γ · α^(−m)
```

Baby-step Giant-step

Complexity

The complexity of baby-step giant-step algorithm is:

memory $\mathcal{O}(\sqrt{n})$ bytes

time $\mathcal{O}(\sqrt{n})$ group multiplications

Pollard's rho

Description

- Let's assume that the group order n is prime.
- Define a partition of $\mathbb{G}$ in three subsets S_1, S_2 and S_3 of roughly equal size. (with $1 \notin S_2$)
- Define a sequence of group elements x_i by:
  - $x_0 = 1$
  - $x_{i+1} = f(x_i) =$
    - $\beta \cdot x_i$ if $x_i \in S_1$
    - x_i^2 if $x_i \in S_2$
    - $\alpha \cdot x_i$ if $x_i \in S_3$
- Each x_i can be written as $x_i = \alpha^{a_i}\beta^{b_i}$ with $a_0 = 0$, $b_0 = 0$ and:
  - $a_{i+1} =$
    - a_i if $x_i \in S_1$
    - $2a_i$ if $x_i \in S_2$
    - $a_i + 1$ if $x_i \in S_3$
  - $b_{i+1} =$
    - $b_i + 1$ if $x_i \in S_1$
    - $2b_i$ if $x_i \in S_2$
    - b_i if $x_i \in S_3$

POLLARD's rho

Description

- Use FLOYD's algorithm to find x_i and x_{2i} such that $x_i = x_{2i}$.
- We then have:

$$\begin{aligned} \alpha^{a_i}\beta^{b_i} &= \alpha^{a_{2i}}\beta^{b_{2i}} \\ \iff \beta^{b_i - b_{2i}} &= \alpha^{a_{2i}-a_i} \end{aligned}$$

- Taking logarithms in base α yields:

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

- Provided that $b_i \not\equiv b_{2i} \pmod{n}$ the solution is given by:

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

Université de Limoges

POLLARD's rho

Description

Hints for the partition of $\mathbb{G}$

The partition of $\mathbb{G}$ in $S_1 \cup S_2 \cup S_3$ can be done as according to the following rule:

- $x \in S_1$ if $x \equiv 1 \pmod{3}$
- $x \in S_2$ if $x \equiv 0 \pmod{3}$
- $x \in S_3$ if $x \equiv 2 \pmod{3}$

Complexity

- The time complexity of POLLARD's rho algorithm is $\mathcal{O}(\sqrt{n})$
- Memory requirement is negligible

Université de Limoges

Pohlig-Hellman

Description

- This method takes advantage of the factorisation of the group order.
- Let $n = p_1^{e_1} p_2^{e_2} \ldots p_r^{e_r}$ be the prime factorisation of n.
- The basic idea is that if $x = \log_\alpha \beta$ then it is possible to compute

$$x_i = x \bmod p_i^{e_i} \qquad \text{for } 1 \leq i \leq r$$

  by taking logarithms modulo $p_i^{e_i}$.
- Considering the p_i-ary representation of x_i:

$$x_i = \ell_{e_i-1} p_i^{e_i-1} + \cdots + \ell_1 p_i + \ell_0$$

  x_i is determined by successively computing the digits $l_0, l_1, \ldots, l_{e_i-1}$ in turn.
- x is finally determined by applying CRT recombination on $(x_1, \ldots, x_r)$.

Université de Limoges

Pohlig-Hellmann

Complexity

Complexity

The complexity of Pohlig-Hellmann algorithm is $\mathcal{O}\left(\Sigma_{i=1}^{r} e_i(\log n + \sqrt{p_i})\right)$ group multiplications.

Use case

The Pohlig-Hellmann method is efficiently applicable if n has only small prime factors.

Université de Limoges

Index-calculus
Description

- Index-calculus is the most efficient method for computing discrete logarithms, but ...
- ...it applies only on particular groups: $\mathbb{Z}_p^\star, \mathbb{F}_{2^m}^\star, \ldots$
- The index-calculus method requires the selection of a (relatively) small subset $S = \{p_1, p_2, \ldots, p_t\}$ of elements of $\mathbb{G}$.

  S is called the factor base and should be such that a random group element can be expressed as a product of elements from S with good probability.
- The most time consuming part is a pre-computation phase aimed at determining logarithms of the factor base elements.

Index-calculus
Algorithm (pre-computation)

Algorithm 2 Index-calculus algorithm

Input: A group $\mathbb{G}$ of order n, $\alpha \in \mathbb{G}$ a generator, $\beta \in \mathbb{G}$

Output: $x = \log_\alpha \beta$

1: Choose a subset $S = \{p_1, p_2, \ldots, p_t\}$ of $\mathbb{G}$ called the factor base.

2: Select a random integer k, $0 \leq k \leq n-1$, and compute α^k.

3: Try to express α^k as a product of elements from S:

$$\alpha^k = \Pi_{i=1}^t p_i^{c_i}, \; c_i \geq 0$$

If successful, takes the logarithms to obtain a linear relation:

$$k \equiv \Sigma_{i=1}^t c_i \log_\alpha p_i \pmod{n}$$

4: Repeat steps 2 and 3 until more than t relations are obtained.

5: Solve mod n the system of linear equations to obtain values of $\log_\alpha p_i$, $1 \leq i \leq t$.

Index-calculus
Description

- The factor base S must be neither too small nor too large:
  - not too small: so that the number of candidates for being a product of elements from S is not prohibitive,
  - not too large: because at some point one have to solve a system with as many equations as the size of S.
- Given $\mathbb{G}$, the pre-computation phase must be performed only once.
- Notice that this pre-computation phase is easily parallelizable.
- The discrete logarithm is computed in a short second step.
- For each discrete logarithm to compute in $\mathbb{G}$, values of $\log_\alpha p_i$ are re-used and only the second step of the algorithm must be executed.

Index-calculus
Algorithm

Algorithm 2 Index-calculus algorithm

Input: A group $\mathbb{G}$ of order n, $\alpha \in \mathbb{G}$ a generator, $\beta \in \mathbb{G}$
Output: $x = \log_\alpha \beta$

1. Choose a subset $S = \{p_1, p_2, \ldots, p_t\}$ of $\mathbb{G}$ called the factor base.
2. Select a random integer k, $0 \leq k \leq n-1$, and compute α^k.
3. Try to express α^k as a product of elements from S:

$$\alpha^k = \Pi_{i=1}^t p_i^{c_i}, \; c_i \geq 0$$

   If successful, takes the logarithms to obtain a linear relation:

$$k \equiv \Sigma_{i=1}^t c_i \log_\alpha p_i \pmod{n}$$

4. Repeat steps 2 and 3 until more than t relations are obtained.
5. Solve mod n the system of linear equations to obtain values of $\log_\alpha p_i$, $1 \leq i \leq t$.
6. Select a random integer k, $0 \leq k \leq n-1$, and compute $\beta\alpha^k$.
7. Try to express $\beta\alpha^k$ as a product of elements from S:

$$\beta\alpha^k = \Pi_{i=1}^t p_i^{d_i}, \; d_i \geq 0$$

   If successful, takes the logarithms in base α to obtain:

$$x = \log_\alpha \beta = \left(\Sigma_{i=1}^t d_i \log_\alpha p_i - k\right) \bmod n$$

Index-calculus

Complexity

Subexponential-time algorithms

Let $0 < \alpha < 1$ and $c > 0$ two real constants:

An algorithm which takes as input an integer q (of size $\ln q$) is said subexponential time if its running time is:

$$L_q[\alpha, c] = \mathcal{O}\left(\exp\left((c + o(1))(\ln q)^\alpha (\ln \ln q)^{1-\alpha}\right)\right) ,$$

- For $\alpha = 0$, $L_q[0, c]$ is a polynomial in the input size $\ln q$.
- For $\alpha = 1$, $L_q[1, c]$ is a polynomial in q, so is exponential in $\ln q$.

The nearest α is to 0, the more the algorithm is polynomial-time like.

The nearest α is to 1, the more the algorithm is exponential-time like.

Index-calculus in $\mathbb{Z}_p^\star$

Complexity

Factor base

- When $\mathbb{G} = \mathbb{Z}_p^\star$ a good choice for S is to select the first t primes.
- A relation is collected each time α^k is p_t-smooth.

Complexity

- Provided that the factor base size t is optimally chosen, the complexity of the presented index-calculus algorithm is $L_n[\frac{1}{2}, c]$ for some constant $c > 0$.
- The fastest known variant of the index-calculs algorithm is called Number Field Sieve and achieves a complexity of $L_n[\frac{1}{3}, 1.923]$.