The discrete logarithm problem.
1 – Generic algorithms

Pierrick Gaudry

Caramel – LORIA
CNRS, Université de Lorraine, Inria

MPRI – 12.2 – 2013-2014

References

Recommended book:

Mathematics of Public Key Cryptography, by Steven Galbraith. Cambridge University Press, 2012.

600 pages, most of them related to 12.2 in terms of topic. Should be at the right difficulty level (not too elementary, not too much mathematical background is assumed).

Plan

Motivation

Square root algorithms

Complexity results

Variants of the DLP

A few words of crypto

Public key cryptography was invented in the 70's.
This solves major practical problems for the deployment of crypto in everyday life:

- Key exchange over an insecure channel;
- Certificates (be sure that you are talking to the right person);
- Signatures.
- ...

The most basic (and historically important) algorithm is the **Diffie-Hellman key exchange**.
All currently used public key algorithms have their security rely on number theoretic problems:

- Integer factorization (RSA);
- Discrete logarithm.

The discrete logarithm problem

Definition: the discrete log problem

Let G be a cyclic group of order N, with a generator g.
The DLP is:

Given $h \in G$, find an integer x such that $h = g^x$.

Classical assumptions:

- The order N is known.
- The group G is effective, i.e. we have
  - a compact representation of the elements of G (ideally, in $O(\log N)$ bits);
  - an efficient algorithm for the group law (polynomial time in $\log N$).

Rem: the integer x makes sense only modulo N.

The DL as an explicit isomorphism

Cyclic group isomorphism

G is a cyclic group means

$$G \cong \mathbb{Z}/N\mathbb{Z}.$$

The map from $\mathbb{Z}/N\mathbb{Z}$ to G is the **exponentiation**:

$$x \mapsto g^x.$$

It can be computed in $O(\log N)$ operations in G.

The inverse map from G to $\mathbb{Z}/N\mathbb{Z}$ is the **DL**:

$$g^x \mapsto x.$$

It is a much harder problem. The exhaustive search will take N operations.

Security – Orders of magnitude

Important question:

How many operations can The Big Computer do ?

Large-scale academic computations (e.g. factorization of RSA-768): about 2^{65} operations. This corresponds to enough energy dissipation to heat to boiling point 2 olympic pools.

Handling 2^{112} operations would correspond to **boiling all the water on the planet**. [Credits: *Universal security*, by Lenstra, Kleinjung, Thomé]

Conclusion: in crypto, we consider as secure a system for which the best attack requires at least 2^{128} operations.

Exercise: read *Digital Fortress* by Dan Brown, and look for numerous mistakes.

Examples of groups

Easy groups:

- $(\mathbb{Z}/N\mathbb{Z}, +)$. The DL is just a division.
- $\{\exp(2i\pi/n) : i \in [0, n-1]\}$. The DL is more or less the classical logarithm.

Moderately hard groups:

- Finite fields of prime order. DLP can be solved in subexponential time.
- Class groups of number fields. DLP can be solved in subexponential time.
- Finite fields of small characteristic. Quasi-polynomial time algorithm (practical ?).

Hard groups:

- Elliptic curves.
- Genus 2 hyperelliptic curves.

Plan

Motivation

Square root algorithms

Complexity results

Variants of the DLP

Pohlig–Hellman reduction: CRT step

Assume that we know the factorization of the group order

$$N = \prod p_i^{e_i}.$$

For each i, let us project $h = g^x$ on the **subgroup** of order $p_i^{e_i}$:
Let

$$g_i = g^{N/p_i^{e_i}} \quad \text{and} \quad h_i = h^{N/p_i^{e_i}}.$$

Then, g_i is of order $p_i^{e_i}$ and

$$h_i = g_i^{x_i}$$

where x_i is congruent to x modulo $p_i^{e_i}$.

Algorithm: Compute x_i for every i, and then reconstruct x by the Chinese Remainder Theorem.

Pohlig–Hellman reduction: Hensel step

From the previous slide, we can assume that $N = p^e$.
Let us write the unknown x in base p:

$$x = x_0 + x_1 p + x_2 p^2 + \cdots + x_{e-1} p^{e-1}.$$

Then, x_0 can be found as the discrete logarithm between

$$g_0 = g^{p^{e-1}} \quad \text{and} \quad h_0 = h^{p^{e-1}},$$

in the **subgroup** of order p.

Induction:
Assuming $x_0, \ldots, x_{i-1}$ are known, we have:

$$h \; g^{-(x_0 + x_1 p + \cdots + x_{i-1} p^{i-1})} = g^{p^i (x_i + x_{i+1} p + \cdots + x_{e-1} p^{e-1-i})}.$$

Raising both sides to the power p^{e-i-1}, we obtain a discrete log equation in the subgroup of order p, with solution x_i.

Pohlig–Hellman reduction: result

Theorem of Pohlig–Hellman

Let G be a cyclic group of known factored order $N = \prod p_i^{e_i}$.
The DLP in G can be solved at the cost of

- For each i, solving e_i instances of DLP in a group of order p_i;
- Additional operations in G (a number that is polynomial in $\log N$).

Rem: Since the number of factors and the exponents are polynomial in $\log N$, sometimes the result is stated as

> The DLP in a cyclic group is not harder than the DLP in its subgroup of largest prime order.

From now, we concentrate on DLP in **prime order groups**.

Shanks' baby-step giant-step algorithm

This algorithm is a **time-memory tradeoff**.
Let K be a parameter (in the end, $K \approx \sqrt{N}$). Write the dlog x as

$$x = x_0 + K\,x_1, \quad \text{with} \quad 0 \le x_0 < K \text{ and } 0 \le x_1 < N/K.$$

Algorithm

1. Compute **Baby Steps**:
   For all i in $[0, K-1]$, compte g^i.
   Store in a hash table the resulting pairs (g^i, i).
2. Compute **Giant Steps**:
   For all j in $[0, \lfloor N/K \rfloor]$, compute hg^{-Kj}.
   If the resulting element is in the BS table, then get the corresponding i, and return $x = i + Kj$.

Important remark: once g^{-K} has been precomputed, each giant step costs only one operation.

Baby-step giant-step algorithm: analysis

Total cost: $K + N/K$ group operations. We do not count the searches in the table.
Memory requirement: K group elements and indices.
Optimize for the worst-case: take $K = \lfloor \sqrt{N} \rfloor$.

Theorem

Discrete logarithms in a cyclic group of order N can be computed in less than $2\lceil \sqrt{N} \rceil$ operations.

If one is interested in the **average** running time, one expect the algorithm to stop at the middle of the giant steps.
Need to choose K for minimizing $K + N/2K$. That is, take $K = \sqrt{N/2}$, and this yields:

Thm. Discrete logarithms in a cyclic group of order N can be computed in expected time $\sqrt{2N}$ group operations.

Baby-step giant-step algorithm: exercises

In the case where the group G is the set of points of an elliptic curve, the inverse in the group is very cheap compared to a group operation.

Exercise 1:

- Adapt the BSGS algorithm to take advantage of this cheap inversion.
- Optimize parameters for a worst-case complexity.
- Optimize parameters for an average-case complexity.

Sometimes, we know that the discrete log lies in a subinterval $[a, b]$ of $[0, N-1]$.

Exercise 2:

- Adapt the BSGS algorithm to this situation.
- Combine with the previous improvement when the inversion is cheap.

Pollard's Rho algorithm

Pollard's Rho algorithm for integer factorization can be adapted to DL computations.

Differences:

- Detection of a match: for IF, have to compute a GCD, for DL, this is just a comparison of elements.
- In IF, we can forget the link with the starting point. In DL, we need something to get the result after a collision has been found.

Need a **pseudo-random function** $f : G \to G$. We construct it as follows:

- For $0 \leq i < 20$, let a_i, b_i be random integers modulo N.
- Precompute $T_i = g^{a_i} h^{b_i}$.
- For an element $z \in G$, we define

  $$f(z) = z\, T_{\mathcal{H}(z)},$$

  where $\mathcal{H}$ is a hash function from G to $[0, 19]$.

Pollard's Rho algorithm

Key feature: if we know a formula $z = g^a h^b$ for z, we can deduce a similar formula for $f(z)$.

Pollard's Rho Algorithm

1. Compute $z_0 = g^{x_0} h^{y_0}$, for random x_0, y_0;
2. Compute the sequence $z_{n+1} = f(z_n)$, and simlutaneously $x_{n+1} = x_n + a_{\mathcal{H}(z_n)}$ and $y_{n+1} = y_n + b_{\mathcal{H}(z_n)}$. At each step, we have $z_n = g^{x_n} h^{y_n}$.
3. If we have a collision $z_n = z_m$ for $n \neq m$, then return

$$x \equiv (x_n - x_m)/(y_m - y_n) \mod N.$$

Rem. For collision detection without storage, use Floyd's algo.
Rem. If $\gcd(y_n - y_m, N) \neq 1$, then the algorithm fails. If N is prime, this occurs with probability $1/N$.

First conclusion on generic algorithms

Combining Pohlig-Hellman and Pollard's Rho, we get:

Solving DLP in generic groups

Let G be a cyclic group of order N, whose largest prime factor is p. Discrete logarithms in G can be computed in $O(\sqrt{p})$ group operations, and $O(1)$ storage.

Rem. The $O(1)$ storage is heuristic. If we allow $O(\sqrt{p})$ elements to be stored, the result is rigorously proven.
We'll see later that this algorithm is optimal, up to a constant factor.

Question: what about parallelization ?

Pollard's Rho with distinguished points

We want an **arbitrary criterion** that says *"this point is special"*.
Let $\mathcal{H}'$ be a hash function that has nothing to do with $\mathcal{H}$.

Definition: distinguished point

An element z of G is k-distinguished if the first k bits of $\mathcal{H}'(z)$ are 0.

Idea: replace Floyd's cycle detection by **storing all distinguised points** that we find in the sequence z_n, and look for a collision among them.
It works, because, if there is a collision between non-distinguished points, the two sequences will continue on the same path and reach sooner or later a distinguished point.
Comparison with the store-all-points strategy:

- Divide by 2^k the memory requirement;
- Add about 2^k steps (delay before detection).

Parallel collision search

Alternative approach: When a distinguished point is hit, start a **new sequence** from scratch.

Parallel algorithm (client / server model):

1. The server precomputes the T_i that define f and send them to the clients.
2. All clients do the same **independently**:
   - 2.1 Pick a starting point $z_0 = g^{x_0} h^{y_0}$ at random;
   - 2.2 Construct the sequence $z_{n+1} = f(z_n)$.
   - 2.3 When z_n is distinguished, it is sent to the server together with its formula in terms of g and h. Then the client goes back to step 2.1.
3. Each time the server receives a distinguished point, it stores it in a datastructure.
4. When the server detects that it has received twice the same point, it computes the discrete log.

Parallel collision search: analysis

The analysis no longer relies on statistics on functional graphs of random functions, but on the **birthday paradox**.

Result:

- This algorithm gives an almost perfect N-fold speed-up if N processors are available.
- The memory required on each client is $O(1)$ group elements.
- The memory required on the server is (say) 1000 times the number of clients.
- It works very well in practice. Used for setting records in ECDLP computations.

Cheap automorphisms

The trick with the inverse map can be genrealized.
Hypothesis: there exists an **automorphism** ϕ of G, such that we can quickly compute a canonical representative for each orbit:

> *Given $z \in G$, compute $\bar{z}$ in $O_z = \{\phi^k(z) : k \in \mathbb{Z}\}$, such that for any $z' \in O_z$, $\bar{z'} = \bar{z}$.*

Rem. If k is small and ϕ can be efficiently computed, then it's ok.
We require furthermore that if z is known in terms of g and h, then so is $\bar{z}$.
Then, modify the sequence z_n, so that it works only with canonical representatives.
Result:

> Let G be a group with a cheap automorphism of order K. Then we can save a factor $\sqrt{K}$ in the discrete log complexity.

Main example: Elliptic curves defined over a subfield.

Plan

Motivation

Square root algorithms

Complexity results

Variants of the DLP

A lower bound

Theorem (Shoup): Lower bound on DLP

Let A be a probabilistic generic algorithm for solving the DLP. If A succeeds with probability at least $\frac{1}{2}$ on a group G, then A must perform at least $\Omega(\sqrt{\#G})$ group operations in G.

Rem. Can be refined, for algorithms with proba of success that is at least $1/(\log G)^k$.

Consequence: Pollard's Rho is optimal, up to constants.

Rem: Previously, Nechaev proved the same result for deterministic algorithms.

What is a generic group?

Def. An encoding of a group G of prime order N is an injective function $\sigma : G \to \{0,1\}^{\lceil t \log N \rceil}$, where t is some constant $t \geq 1$.

Def. A generic algorithm A for the DLP in a group G of order N, takes as input N, $\sigma(g)$, $\sigma(h)$, returns an integer x, and access to an **oracle** O:
When the oracle O is invoked with two inputs $\sigma(z_1)$ and $\sigma(z_2)$, it returns $\sigma(z_1/z_2)$.

The algorithm A succeeds if x verifies $h = g^x$.

Tools for the proof

Main idea: the oracle O will play hide-and-seek with the algorithm A.
The algorithm is not given the encoding map σ. The oracle will change it (define it!) on-the-fly.
It has to ensure that it gives **consistent ouput** to A.
As long as the number of queries stays low, the oracle has enough room to play.

Modelisation: the "constraints" can be encoded inside the oracle with multivariate polynomials.

Key Lemma: Let $F(x_1, \ldots, x_k) \in \mathbb{F}_N[x_1, \ldots, x_k]$ be a non-zero polynomial of total degree d. Then for $(x_1, \ldots, x_k)$ chosen uniformly at random in $\mathbb{F}_N^k$, the probability that $F(x_1, \ldots, x_k) = 0$ is at most d/N.

Random self-reducibility

Hypothesis: we have an algorithm A that solves the discrete logarithm problem in G for a fraction $1/P$ of the inputs.

Let h be an element for which we want the discrete log.
Repeat:

- Pick a random integer a;
- Compute $h' = hg^a$, and try to solve the DLP for h' with A.
- If it works, then $\log h = \log h' - a$.

One expects that after P trials, it works.

In complexity theory, this property is called **random self-reducibility**.

Plan

Motivation

Square root algorithms

Complexity results

Variants of the DLP

The DH problem

Remember the DH protocol? What is its security?

The Diffie-Hellman Problem

Let G be a cyclic group. Given g, g^a, g^b three elements of G, the DHP is to compute g^{ab}.

The DHP can not be harder than the DLP (if we know how to compute DL, then one computes a from g^a and raise g^b to the power a).

Question: converse ?

Rem. There exists a decisional variant of the DHP: given g, g^a, g^b, g^c, try to guess if $c = ab$. Very useful in crypto.

Maurer–Wolf reduction

Theorem (Maurer–Wolf). Reduction DHP–DLP

Let G be a group of order N. Assume that there exists an algorithm A that can solve the (computational) DHP. Then, there exists an algorithm that can solve the DLP in G, that will do a subexponential number of calls to A.

By subexponential, we mean:

- $L_N(1/2)$ if we accept a reasonable heuristic related to elliptic curves.
- $L_N(2/3)$ for a proven result (with hyperelliptic curves).

Conclusion: some kind of “proof” that DHP is hard, if one believes that DLP is hard in our specific instance of G.

Low Hamming weight DLP

If we know in advance that the discrete log (as an integer in $[0, N-1]$) has a low Hamming weight w in base 2, then we can use it to speed-up the computations.

Theorem (Coppersmith, Stinson). Low Hamming weight DLP

Let G be a group of order N, and set $n = \log_2 N$. There exists an algorithm that can compute a DLP in G of weight w in $O(\sqrt{w}\binom{n/2}{w/2})$ operations in G, with a storage requirement of $O(\sqrt{w}\binom{n/2}{w/2})$ elements.

This is about the square root of the search space, so it fits within the general "square root improvements".

Rem. Memory requirement is huge. Still open to get a full square-root improvement with low-memory.