

# MPRI – Cours 2.12.2



F. Morain



## Lecture II: discrete logarithm in generic groups

2012/10/15

The slides are available on <http://www.lix.polytechnique.fr/Labo/Francois.Morain/MPRI/2012>

# Contents

- I. Introduction.
- II. The Pohlig-Hellman reduction.
- III. Baby steps giant steps.
- IV. Pollard's  $\rho$ .
- V. Nechaev/Shoup theorem (à la Stinson).

## I. Introduction

**Def.** (DLP) Given  $G = \langle g \rangle$  of order  $n$  and  $a \in G$ , find  $x \in [0..n[$  s.t.  $a = g^x$ .

**Adaptive and non-adaptive:**  $a$  is given beforehand, or only after some precomputation have been done (see Adleman's algorithm later).

**Goal:** find a **resistant** group.

**Rem.** DL is easy in  $(\mathbb{Z}/N\mathbb{Z}, +)$ , since  $a = xg \bmod N$  is solvable in polynomial time (Euclid).

**Relatively easy groups:** (subexponential methods) finite fields, curves of very large genus, class groups of number fields.

**Probably difficult groups:** (exponential methods only?) elliptic curves.

## Variants of the DL problem

**Decisional DH problem:** given  $(g, g^a, g^b, g^c)$ , do we have  $c = ab \bmod n$ ?

**Computational DH problem:** given  $(g, g^a, g^b)$ , compute  $g^{ab}$ .

**DL problem:** given  $(g, g^a)$ , find  $a$ .

**Prop.** DL  $\Rightarrow$  CDH  $\Rightarrow$  DCDH.

**Thm.** converse true for a large class of groups (Maurer & Wolf – see Smith's part).

**More problems:**  $\ell$ -SDH (given  $g, g^\alpha, \dots, g^{\alpha^\ell}$ , compute  $g^{\alpha^{\ell+1}}$ ).

**Rem.** Generalized problems on pairings.

## Generic groups

**Rem. generic** means we cannot use specific properties of  $G$ , just group operations.

**Known generic solutions:**

- enumeration:  $O(n)$ ;
- Shanks: deterministic time and space  $O(\sqrt{n})$ ;
- Pollard: probabilistic time  $O(\sqrt{n})$ , space  $O(1)$  elements of  $G$ .

**Rem.** All these algorithms can be more or less distributed.

## II. The Pohlig-Hellman reduction

**Idea:** reduce the problem to the case  $n$  prime.

$$n = \prod_i p_i^{\alpha_i}$$

Solving  $g^x = a$  is equivalent to knowing  $x \bmod n$ , i.e.  $x \bmod p_i^{\alpha_i}$  for all  $i$  (chinese remainder theorem).

**Idea:** let  $p^\alpha \parallel n$  and  $m = n/p^\alpha$ . Then  $b = a^m$  is in the cyclic group of order  $p^\alpha$  generated by  $g^m$ . We can find the log of  $b$  in this group, which yields  $x \bmod p^\alpha$ .

**Cost:**  $O(\max(DL(p^\alpha))) = O(\max(DL(p)))$ .

**Consequence:** in DH,  $n$  must have at least one large prime factor.

## III. Baby steps giant steps (1/2)

Shanks:

$$x = cu + d, 0 \leq d < u, \quad 0 \leq c < n/u$$

$$g^x = a \Leftrightarrow a(g^{-u})^c = g^d.$$

**Step 1 (baby steps):** compute  $\mathcal{B} = \{g^d, 0 \leq d < u\}$ ;

**Step 2 (giant steps):**

- compute  $f = g^{-u} = 1/g^u$ ;
- $h = a$ ;
- for  $c = 0..n/u$   
 {will contain  $af^c$ }  
 if  $h \in \mathcal{B}$  then stop; else  $h = h \cdot f$ .

**End:**  $h = af^c = g^d$  hence  $x = cu + d$ .

**Number of group operations:**  $C_o = u + n/u$ , minimized for  $u = \sqrt{n}$ .

## Shanks (2/2)

In the worst case, Step 2 requires  $n/u$  membership tests.

$\mathcal{B}$	insertions	membership tests
list	$u \times O(1)$	$\frac{n}{u} O(u)$
sorted	$O(u \log u)$	$\frac{n}{u} O(\log u)$
hash table	$u \times O(1)$	$\frac{n}{u} O(1)$

**Prop.** If membership test =  $O(1)$ , then dominant term is  $C_o$ , minimal for  $u = \sqrt{n} \Rightarrow$  (deterministic) time and space  $O(\sqrt{n})$ .

**Rem.** all kinds of trade-offs possible if low memory available.



Let  $X_i = a^{c_i} g^{d_i}$ , where  $c_0 = 1, d_0 = 0$ , and  $c_{i+1}, d_{i+1}$  obtained via use of  $f$ .

When  $e$  is found:

$$a^{c_{2e}} g^{d_{2e}} = a^{c_e} g^{d_e}$$

or

$$a^{c_{2e}-c_e} = g^{d_e-d_{2e}}$$

i.e.,

$$x(c_{2e} - c_e) \equiv (d_e - d_{2e}) \pmod{n}.$$

With high probability,  $c_{2e} - c_e$  is invertible modulo  $n$ ; otherwise, we have a factor of  $n$ ...

## Storing a few points:

- Compute  $r$  random points  $M_k = g^{\gamma_k} h^{\delta_k}$  for  $1 \leq k \leq r$ ;
- use  $\mathcal{H} : G \rightarrow \{1, \dots, r\}$ ;
- define  $F(Y) = Y \cdot M_{\mathcal{H}(Y)}$ .

Experimentally,  $r = 20$  is enough to have a large mixing of points. Under a plausible model, this leads to a  $O(\sqrt{n})$  method (see Teske).

**Storing a lot of points:** (van Oorschot and Wiener) using **distinguished point** (say  $x \cdots y000000$ ). Key to a distributed implementation.

## V. Nechaev/Shoup theorem (à la Stinson)

**Encoding function:** injective map  $\sigma : \mathbb{Z}/n\mathbb{Z} \rightarrow S$  where  $S$  is a set of binary strings s.t.  $\#S \geq n$ .

**Ex.**  $G = (\mathbb{Z}/q\mathbb{Z})^* = \langle g \rangle, n = q - 1, \sigma : x \mapsto g^x \pmod{q}, S$  can be  $\{0, 1\}^\ell$  where  $q < 2^\ell$ .

**Wanted:** a **generic algorithm** should work for any  $\sigma$ , in other words it receives  $\sigma$  as an input.

**Oracle  $\mathcal{O}$ :** given  $\sigma(i)$  and  $\sigma(j)$ , computes  $\sigma(ci \pm dj \pmod{n})$  for any given known integers  $c$  and  $d$ . **This is the only operation permitted.**

**Game:** given  $\sigma_1 = \sigma(1)$  and  $\sigma_2 = \sigma(x)$  for random  $x$ , GENLOG succeeds if it outputs  $x$ .

**Ex.** Pollard's algorithm belongs to this class.

## Stinson (2/5)

GENLOG produces  $(\sigma_1, \sigma_2, \dots, \sigma_m)$  using  $\mathcal{O}$  where

$$\sigma_i = \sigma(c_i + xd_i \pmod{n}),$$

with  $(c_1, d_1) = (1, 0)$  and  $(c_2, d_2) = (0, 1), (c_i, d_i) \in \mathbb{Z}/n\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z}$ .

**Two cases:** non-adaptive (choose  $c_i, d_i$  before receiving  $\sigma(x)$ ) or adaptive.

**Thm.** Let  $\beta = \text{Proba}(\text{GenLog succeeds})$ . For  $\beta > \delta > 0$ , one must have  $m = \Omega(n^{1/2})$ .

**The non-adaptive case:****Step 1:** (precomputations) GenLog chooses

$$\mathcal{C} = \{(c_i, d_i), 1 \leq i \leq m\} \subset \mathbb{Z}/n\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z}$$

**Step 2:** upon receiving  $\sigma(x)$ , computes all  $\sigma_i = \sigma(c_i + xd_i)$ .**Step 3:** check whether  $\sigma_i = \sigma_j$  for some  $(i, j)$ ; since  $\sigma$  is injective,  $\sigma_i = \sigma_j$  iff  $c_i + xd_i \equiv c_j + xd_j$ , return  $x$ .**Step 4:** return a random value  $y$ .**Analysis:**

$$\text{Good}(\mathcal{C}) = \{(c_i - c_j)/(d_i - d_j)\}, \#\text{Good}(\mathcal{C}) = \mathcal{G} \leq m(m-1)/2.$$

If  $x \in \text{Good}(\mathcal{C})$ , GenLog returns  $x$ , otherwise some  $y$ . $\alpha$  is the event “ $x \in \text{Good}(\mathcal{C})$ ”:

$$\begin{aligned} \text{Proba}(\beta) &= \text{Proba}(\beta|\alpha)\text{Proba}(\alpha) + \text{Proba}(\beta|\bar{\alpha})\text{Proba}(\bar{\alpha}) \\ &= 1 \times \frac{\mathcal{G}}{n} + \frac{1}{n-\mathcal{G}} \times \frac{n-\mathcal{G}}{n} \\ &= \frac{\mathcal{G}+1}{n} \leq \frac{m(m-1)/2+1}{n}. \end{aligned}$$

 $\Rightarrow$  if  $\text{proba} > \delta > 0$ , then  $m$  must be  $\Omega(n^{1/2})$ .  $\square$ **The adaptive case:** For  $1 \leq i \leq m$ ,  $\mathcal{C}_i = \{\sigma_j, 1 \leq j \leq i\}$ . Then  $a$  can be computed at time  $i$  if  $a \in \text{Good}(\mathcal{C}_i)$ . If  $a \notin \text{Good}(\mathcal{C}_i)$ , then  $a \in \mathbb{Z}/n\mathbb{Z} - \text{Good}(\mathcal{C}_i)$  with  $\text{proba } 1/(n - \#\text{Good}(\mathcal{C}_i))$ .**And now, what?** this result tells you (**only**) that if you want an algorithm that is faster than Pollard’s  $\rho$  or Shanks, then you have to work harder...