Introduction to mathematical cryptography

Lecture 2: Analysing the Discrete Logarithm Problem (DLP)

Sabrina Kunzweiler

Preliminary Arizona Winter School 2025

Inria
The University of Arizona
Institut de Mathématiques de Bordeaux

The discrete logarithm problem (recap)

Discrete Logarithm Problem (DLP)

For $g \in \mathbb{F}_p^*$ primitive element and $A \in \mathbb{F}_p^*$,
the DLP asks to find $a \in \mathbb{Z}$ so that $\exp_g(a) = A$.
Notation: $a = \mathrm{dlog}_g(A)$.

- Security of the Diffie-Hellman key exchange protocol is based on the difficulty of solving DLP.
- Morally, the hardness of DLP depends on the order of g.
- DLP becomes easier when $ord(g)$ is composite (Pohlig-Hellman) $\Rightarrow$ we work in a prime-order subgroup $\langle g \rangle \subset \mathbb{F}_p^*$ (with g not primitive, but $ord(g) = q$ prime).

Linear search (naive algorithm)

Algorithm 0 Linear search

Input: $g \in \mathbb{F}_p^*$ with $ord(g) = q$, and $A \in \langle g \rangle$

Output: $a = \mathrm{dlog}_g(A)$

1: $x \leftarrow 1$
2: **for** $i = 1, \ldots, q$ **do**
3:     $x \leftarrow g \cdot x$
4:     **if** $A = x$ **then**
5:         **return** i
6:     **end if**
7: **end for**

Correctness

- In Line 4, at step i: $x = g^i$, so $A = x \Leftrightarrow i = \mathrm{dlog}_g(A)$.

$\Rightarrow$ Algorithm 0 terminates and the output is correct.

Runtime

(in number of multiplications M)

- Best case: $a = 1 \Rightarrow 1\,\mathrm{M}$
- Worst case: $a = q \Rightarrow q\,\mathrm{M}$
- big-O notation:[a] $O(q)$

Memory $O(1)$

[a] $f(n) = O(h(n))$ if $\exists\, c > 0, n_0 \in \mathbb{N}: |f(n)| \leq c|h(n)|\ \forall\, n > n_0$.

Baby-step giant-step algorithm

BSGS algorithm (informal)

Baby-step giant-step algorithm (BSGS) by David Shanks (1971)

Idea: decompose the solution $a = \mathrm{dlog}_g(A)$ as

$$a = jm + i, \quad \text{with } m = \lfloor \sqrt{q} \rfloor + 1,\ i, j \in \{0, \dots, m-1\}.$$

- baby steps:

$$g_0 = 1,\ g_1 = g,\ g_2 = g^2,\ \dots,\ g_{m-1} = g^{m-1}.$$

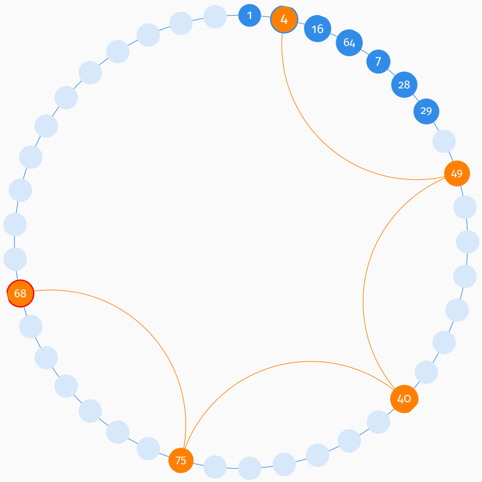
- giant steps:

$$A_0 = A,\ A_1 = g^{-m} \cdot A,\ A_2 = \left(g^{-m}\right)^2 \cdot A,\ \dots,\ \left(g^{-m}\right)^{m-1} \cdot A.$$

- A match $g_i = A_j$ provides us with the solution $a = jm + i$.

Example

Challenge find $a = \mathrm{dlog}_g(A)$ where $g = 4 \in \mathbb{F}_{83}^*$, $A = 68 \in \langle g \rangle$.



Precomputations:
$m = \lfloor \sqrt{41} \rfloor + 1 = 7$,
$h = g^{-7} = 78$.

baby steps

1, 4 (4), 16, 64, 7, 28 29.

giant steps

68, 75, 40, 49, (4), ...

Formal description

Algorithm 1 Shank's BSGS algorithm

Input: $g \in \mathbb{F}_p^*$ with $ord(g) = q$, $A \in \langle g \rangle$
Output: $a = \mathrm{dlog}_g(A)$

$m \leftarrow \lfloor \sqrt{q} \rfloor + 1$
$g_0 \leftarrow 1$
for $i = 1, \ldots, m-1$ **do**
  $g_i \leftarrow g \cdot g_{i-1}$
end for
$\tilde{A} \leftarrow A$
for $j = 0, \ldots, m-1$ **do**
  if $\tilde{A} = g_i$ for some i **then**
    return $jm + i \pmod{q}$
  else
    $\tilde{A} \leftarrow \tilde{A} \cdot g^{-m}$
  end if
end for

Correctness

✓ For $a \in \{1, \ldots, q-1\}$, $\exists\, 0 \leq i, j \leq m-1$ with $a = jm + i$.

Runtime

(in $\mathbb{F}_p$ multiplications M)

- baby steps: $(m-1)\,\mathtt{M}$
- giant steps: $\leq (m-1)\,\mathtt{M}$
- **total** $O(m) = O(\sqrt{q})$

Memory

(in $\mathbb{F}_p$ elements)

- list of baby steps: $O(\sqrt{q})$

Time-memory trade-offs

Having enough memory to store $O(\sqrt{q})$ $\mathbb{F}_p$-elements memory is unrealistic for a “real-world DLP” challenge.

⇒ **Variant of BSGS with memory restrictions**

- Available storage: $m \ll \sqrt{q}$ (finite field elements)
- baby steps: $g_0 = 1, g_1 = g, \ldots, g_{m-1} = g^{m-1}$
- giant steps: $A_0 = A, A_1 = g^{-m} \cdot A, \ldots, A_{\lfloor q/m \rfloor} = \left(g^{-m}\right)^{\lfloor q/m \rfloor}$

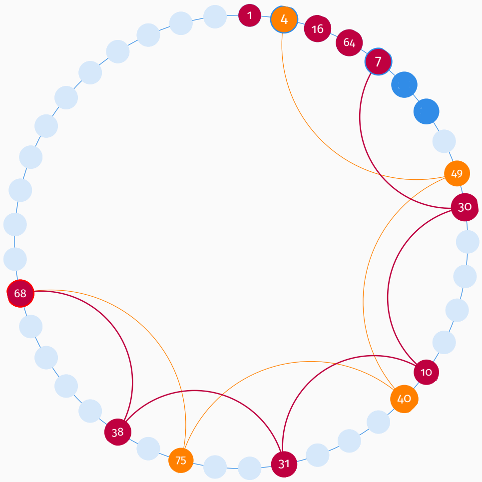
- Runtime: $O(q/m)$ $\mathbb{F}_p$-multiplications
- Memory: $O(m)$ $\mathbb{F}_p$-elements

Remarks

- If $m = \sqrt{q}$, then $O(q/m) = O(\sqrt{q})$ (as in the standard BSGS algorithm).
- *If we allow $m > \sqrt{q}$, is the above variant of BSGS faster?*
  - ⚠ The runtime $O(q/m)$ is only correct if $m \leq \sqrt{q}$.
  - A different (!) variant can provide a time-memory trade-off in the other direction (see exercises).

Example (with time-memory trade-off)

Challenge find $a = \mathrm{dlog}_g(A)$ where $g = 4 \in \mathbb{F}_{83}^*$, $A = 68 \in \langle g \rangle$.



Standard BSGS:
$m = \lfloor \sqrt{41} \rfloor + 1 = 7$,
$h = g^{-7} = 78$.

Time-memory trade-off:
$m = 5$, $h = g^{-5} = 3$

Pollard's rho algorithm

Pollard's rho algorithm (informal)

Pollard's rho algorithm suggested by John M. Pollard (1978)

Idea to find $a = \mathrm{dlog}_g(A)$.
Create a sequence

$$x_0 = g^0 A^0,\ x_1 = g^{k_1} A^{\ell_1},\ x_2 = g^{k_2} A^{\ell_2},\ \ldots$$

$\Rightarrow$ Collision in the sequence $x_i = x_j$ yields a solution:

$$g^{k_i} A^{\ell_i} = g^{k_j} A^{\ell_j} \quad \Leftrightarrow \quad A^{\ell_i - \ell_j} = g^{k_j - k_i}$$

so $a = (k_j - k_i)/(\ell_i - \ell_j)$ if $\ell_i \neq \ell_j \pmod q$.

How to construct a suitable sequence?

- Given x_i, we need to be able to compute (or already know) ℓ_i, k_i.
- The sequence should look random. Then the birthday paradox tells us that a collision occurs after $O(\sqrt{q})$ steps.

A pseudorandom sequence

Consider $f : \mathbb{F}_p^* \to \mathbb{F}_p^*$ defined by

$$x = g^k A^\ell \quad \mapsto \quad x' = \begin{cases} g^k A^{\ell+1} & \text{if } 0 < x < p/3, \\ g^{2k} A^{2\ell} & \text{if } p/3 < x < 2p/3, \\ g^{k+1} A^\ell & \text{if } 2p/3 < x < p. \end{cases}$$

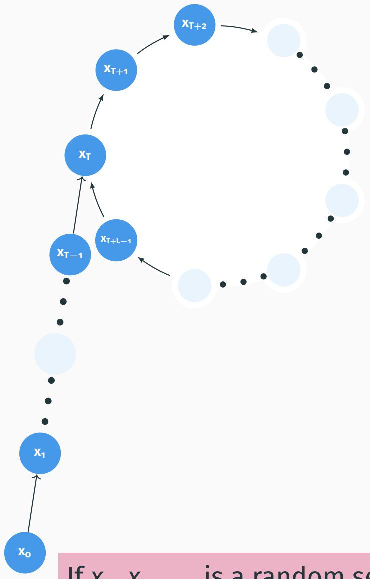
Slight abuse of notation: $f : \mathbb{F}_p^* \times \mathbb{Z}/q\mathbb{Z} \times \mathbb{Z}/q\mathbb{Z} \to \mathbb{F}_p^* \times \mathbb{Z}/q\mathbb{Z} \times \mathbb{Z}/q\mathbb{Z}$, i.e. $(x, k, \ell) \mapsto (x', k', \ell')$ to keep track of the exponents.

Pseudorandom sequence

$$x_0 = g^0 A^0,\ x_1 = f(x_0) = g^{k_1} A^{\ell_1},\ x_2 = f(x_1) = g^{k_2} A^{\ell_2},\ \ldots$$

- This sequence is periodic, since $\mathbb{F}_p^*$ is finite.
- It enters into a loop at some point.

Illustration of the Pollard's rho

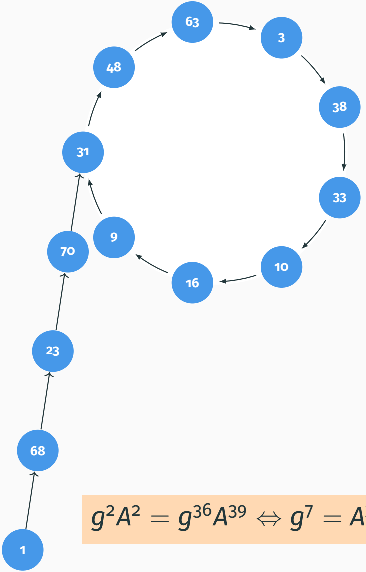


- Tail $x_0, \ldots, x_T$ of tail length T
- Loop $x_T, \ldots, x_{T+L-1}, x_{T+L} = x_T$ of loop length L
- **Birthday paradox** In a class of 30 students, what's the probability that at least two of them share the same birthday?
  $\rightarrow 1 - \prod_{i=1}^{29}(1 - \frac{i}{365}) \approx 71\%$

$\Rightarrow$ **Rule of thumb**: Drawing elements at random from a set of size N, we expect a collision after $\sqrt{\pi N/2}$ draws.

If $x_0, x_1, \ldots$ is a random sequence, we expect $T + L \approx \sqrt{\pi q/2}$.

Example of a Pollard's rho



Compute $\mathrm{dlog}_g(A)$ for $g = 4 \in \mathbb{F}_{83}^*$, $A = 68 \in \langle g \rangle$ (recall $q = 41$)

- $x_0 = g^0 A^0 = 1$
- $x_1 = g^0 A^1 = 68$
- $x_2 = g^1 A^1 = 23$

$\ldots$

- $x_4 = g^2 A^2 = 31$

$\ldots$

- $x_{13} = g^{36} A^{39} = 31$

$T = 4$ (for $x_1, \ldots, x_4$)

$L = 9$ (for $x_4, \ldots, x_{13}$)

$g^2 A^2 = g^{36} A^{39} \Leftrightarrow g^7 = A^{37}$, so $\mathrm{dlog}_g(A) = 7 \cdot 37^{-1} = 29$

Detecting collisions

Given a sequence $x_0, x_1, \ldots$ with tail length T and loop length L, find a collision $x_i = x_j$.

- **Naive strategy** Compute and store $x_0, x_1, x_2, \ldots$ until a collision occurs.
  - runtime: $L + T \approx \sqrt{\pi q/2}$ $\mathbb{F}_p^*$-multiplications
  - memory: $L + T \approx \sqrt{\pi q/2}$ $\mathbb{F}_p$-elements
- **Memory-less variant** Compute (x_i, x_{2i}) for $i = 1, 2, \ldots$ until $x_i = x_{2i}$ for some i.
  - memory: constant (sequence elements are not stored)
  - runtime: $\leq 3 \cdot (T + L) \approx 3\sqrt{\pi q/2}$ $\mathbb{F}_p^*$-multiplications

    Proof idea: We have $x_i = x_{L+i}$ for all $i \geq T$, hence

    $$x_i = x_{2i} \quad \Leftrightarrow i \equiv 2i \pmod{L} \text{ and } i \geq T.$$

    There is precisely one value $T \leq i < T + L$ satisfying these conditions.

Pollard's rho algorithm

Algorithm 2 Pollard's rho algorithm

Input: $g \in \mathbb{F}_p^*$ with $ord(g) = q$, $A \in \langle g \rangle$
Output: $a = \text{dlog}_g(A)$

```
1:  (x, k, ℓ) = (1, 0, 0)
2:  (x', k', ℓ') = (x, k, ℓ)
3:  while True do
4:      (x, k, ℓ) ← f(x, k, ℓ)                          ▷ x = x_i
5:      (x', k', ℓ') ← f(x', k', ℓ')
6:      (x', k', ℓ') ← f(x', k', ℓ')                    ▷ x' = x_{2i}
7:      if x' = x then
8:          if gcd(ℓ' − ℓ, q) = 1 then
9:              return (k − k')(ℓ' − ℓ)^{−1} (mod q)
10:         else
11:             go back to 1 (and change x_0)
12:         end if
13:     end if
14: end while
```

Correctness

✓ If the algorithm terminates (Line 9), the output is correct.

Runtime

(in $\mathbb{F}_p^*$-multiplications)

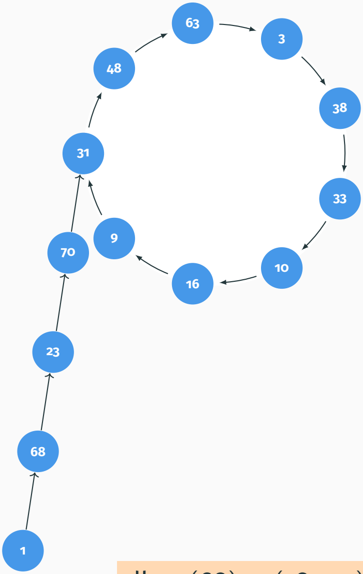
- If f is sufficiently random, then the runtime is expected to be $O(\sqrt{q})$.

Memory

(in $\mathbb{F}_p$ elements)

- only the current sequence elements (x_i, x_{2i}): $O(1)$

Example (Pollard's rho algorithm)



Compute $\mathrm{dlog}_g(A)$ for $g = 4 \in \mathbb{F}_{83}^*$, $A = 68 \in \langle g \rangle$, recall $q = 41$.
(Example from Slide 11)

- $T = 4$
- $L = 9$

$\rightarrow$ expected collision $x_i = x_{2i}$: $4 \leq i < 13$ with $i \equiv 2i \pmod 9$. $\Rightarrow i = 9$.

- We compute $(x_9, k_9, \ell_9) = (33, 18, 18)$ and $(x_{18}, k_{18}, \ell_{18}) = (33, 3, 27)$, and find

$$\mathrm{dlog}_4(68) = (18 - 3) \cdot (27 - 9)^{-1} \equiv 29 \pmod{41}.$$

Index calculus

Index calculus (informal)

Index calculus method going back to Maurice Kraitchik (1922)

Main ingredient Lift $x \in \mathbb{F}_p^*$ to $\hat{x} \in \{1, \ldots, p-1\} \subset \mathbb{Z}$

Setup (to find $a = \mathrm{dlog}_g(A)$)

- Set $\mathcal{P}_B = \{p_1, \ldots, p_b\}$ primes $p_i \leq B$ for some smoothness bound B. Denote $x_i = \mathrm{dlog}_g p_i$

Phase 1: Relation creation

- Find $b+1$ relations among $x_1, \ldots, x_b$ and a, i.e.relations of the form

$$e_1 x_1 + \ldots e_b x_b + a \equiv e \pmod{p-1}.$$

Phase 2: Linear algebra over $\mathbb{Z}/(p-1)\mathbb{Z}$.

- Solve the system of $b+1$ relations in $x_1, \ldots x_b$ and a

Phase 1: Relation creation

Goal Find $b+1$ relations among $x_1, \ldots, x_b$ and a with $x_i = \mathrm{dlog}_g(p_i)$.

Set $\mathcal{R} = \{\}$, until $\#\mathcal{R} = b+1$, repeat the following:

(1) Let $e \in \{1, \ldots, p-1\}$ random.

(2) Compute the lift $z = \widehat{g^e/A} \in \{1, \ldots, p-1\} \subset \mathbb{Z}$

(3) If z is B-smooth, i.e. if $z = \prod_{i=1}^{b} p_i^{e_i}$ for some e_i:
add R_e : $e \equiv e_1 x_1 + \cdots + e_b x_b + a$ to $\mathcal{R}$.

number of primes p_i:
$b = \pi(B) \approx B/\log(B)$
(Prime Number Theorem by Hadamard and de la Vallée Poussin, 1896)

Testing B**-smoothness**:
$O(b)$, for example using trial division[a]

Number of iterations N:
$N \approx u^u$, where $u = \log p / \log B$
(Dickman–de Bruijn function, proportion of smooth number)

[a] There are faster methods like the elliptic curve method (ECM).

Phase 2: Linear algebra, and overall runtime

Phase 2: A system of linear equations over $\mathbb{Z}/(p-1)\mathbb{Z}$

$$\begin{cases} e = e_1 x_1 + \dots e_b x_b + a \\ \qquad \vdots \\ e' = e'_1 x_1 + \dots e'_b x_b + a \end{cases}$$

can be solved in $\tilde{O}(b^3)$, or even $\tilde{O}(b^2)$ using special properties of the system.

$\Rightarrow$ time complexity is neglible compared to Phase 1

Overall complexity

- Dependence on B: time for smoothness testing increases while number of iterations decreases for increasing B.

- Optimal choice: $B \approx e^{1/2\sqrt{\log p \log \log p}}$ (in our setting)

$\Rightarrow$ Runtime: $O\left(e^{2\sqrt{\log p \log \log p}}\right)$ **subexponential**.

Example (Index calculus)

Compute $\mathrm{dlog}_g(A)$ for $g = 4 \in \mathbb{F}_{83}^*$, $A = 68 \in \langle g \rangle$, recall $q = 41$.

Factor base $\mathcal{P}_B = \{2, 3, 5\}$, i.e. $b = 3$.

Phase 1 Choose random exponents $e \in \{1, \ldots, 82\}$, and check for smoothness.

e	$\widehat{g^e/A}$	e	$\widehat{g^e/A}$
15	5^2	31	2^4
59	x	33	x
60	3^2	17	x
7	x	(72)	(2^4)
40	x	43	$2 \cdot 5$

Phase 2

$$\begin{cases} 15 = & & & 2x_5 + a \\ 60 = & & 2x_3 & + a \\ 31 = & 4x_2 & & + a \\ 43 = & x_2 & + & x_5 + a \end{cases} \Rightarrow \quad a = 29 \equiv 70 \pmod{41}$$

(here: $x_2 = \mathrm{dlog}_4(2)$, $x_3 = \mathrm{dlog}_4(3)$, $x_5 = \mathrm{dlog}_4(5)$)

Diffie-Hellman in generic groups

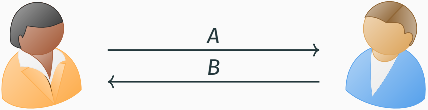
Generalization of the Diffie-Hellman protocol

Let $(\mathbb{G}, \circ)$ be a <u>commutative</u> group.

For $g \in \mathbb{G},\ a \in \mathbb{Z}$ define

$$\exp_g(a) = \underbrace{g \circ \cdots \circ g}_{a \text{ times}}.$$

Generalized Diffie-Hellman protocol



Alice	Bob
$(a \in \mathbb{Z})$	$(b \in \mathbb{Z})$
$A \leftarrow \exp_g(a)$	$B \leftarrow \exp_g(b)$
$K_{AB} \leftarrow \exp_B(a)$	$K_{BA} \leftarrow \exp_A(b)$

Group Discrete Logarithm Problem (Group-DLP)

For $g \in \mathbb{G}$, $A \in \langle g \rangle$ the <u>Group-DLP</u> asks to find $a \in \mathbb{Z}$ so that $\exp_g(a) = A$. Notation: $a = \mathrm{dlog}_g(A)$.

How hard is the Group-DLP?

Generic algorithms

- Baby-step giant-step algorithm, Pollard's rho algorithm:
  They can be applied to any group, no special property of finite fields are used.

$\Rightarrow$ Group-DLP can be solved in $O(\sqrt{N})$, where $N = \#\mathbb{G}$.
This is best possible in a generic group with N prime (Shoup, 97)

Non-generic algorithms

- Index calculus: This algorithm relies on working in finite fields (using lifts to $\mathbb{Z}$ and factorization). It cannot be translated to arbitrary groups.

Next lecture

- Elliptic curves: Groups that are closer to "generic groups"