Crypto Engineering (GBX9SY03)
Memoryless generic discrete logarithm computation in an interval using kangaroos

2022-01-05

Grading

This TP is graded as part of the *contrôle continu*. You must send a written report (in a portable format) detailing your answers to the questions, and the corresponding source code, *including all tests*, **with compilation and execution instructions** by 2022-01-21T18:00+0100) to:

pierre.karpman@univ-grenoble-alpes.fr.

Working in teams of two is allowed and encouraged (but not mandatory), in which case only one report needs to be sent, with the name of both students clearly mentioned.

Introduction

The goal of this exercise is to write a simple implementation of Pollard's Kangaroo algorithm to compute the discrete logarithm of a group element whose exponent is known to lie in a "small" interval, without using (much) memory.

Let $\mathbb{G} = \langle g \rangle$ be a finite group of order N, and $h = g^a$, $a \in [\![0, W]\!]$, $W \ll N$, be an element for which we want to compute the discrete logarithm a. The algorithm is based on the sequence of jumps of two kangaroos: a *tame* kangaroo, that always knows the discrete logarithm of the element it lands on; and a *wild* kangaroo, that can only remember the jumps from its starting point.

Both kangaroos jump deterministically and identically from one group element to another; in other words, both use the same jump map $\mathcal{J} : \mathbb{G} \to \mathbb{G}$. They also regularly lay traps to try to catch each other, and it is clear that if one jumps on an element already visited by the other, the former will eventually get caught by a trap of the latter (in other words, barring a full cycle in the entire group, a kangaroo cannot get caught as long as it is leading (i.e. has the largest logarithm)). When this happens, one has in fact recovered enough information to compute the discrete logarithm of h.

In more details one does the following, where k, μ, d are parameters whose values are to be determined later.

— Split $\mathbb{G}$ into k subsets $\mathcal{S}_j$ of approximately equal size; pick k exponents e_j s.t. their average $1/k \sum_{j=1}^{k} e_j \approx \mu$; define $\mathcal{J}$ from the k partial maps $\mathcal{J}_j : \mathcal{S}_j \to \mathbb{G}$, $x \mapsto xg^{e_j}$.

— The tame kangaroo's sequence (x_n) is defined as $x_0 = g^{\lceil W/2 \rfloor}$ (i.e. the middle of the interval); $x_{i+1} = \mathcal{J}(x_i)$. Notice that at any time the discrete logarithm b_i of $x_i = g^{b_i}$ is known.

— The wild kangaroo's sequence (y_n) is defined as $y_0 = h$; $y_{i+1} = \mathcal{J}(y_i)$. Notice that at any time, one can write y_i as hg^{c_i} where c_i is known.

— Define $\mathcal{D} : \mathbb{G} \to \{0, 1\}$ so that $\Pr[\mathcal{D}(x) = 1 : x \twoheadleftarrow \mathbb{G}] = p$ which returns 1 if its argument is a *distinguished* element.

— Anytime a tame (resp. wild) kangaroo lands on a distinguished element x_i (resp. y_i), it lays a trap by recording (x_i, b_i) (resp. (y_i, c_i)) in an efficient data structure for sets. However, if a trap (y_j, c_j) (resp. (x_j, b_j)) was already present, it instead gets trapped and returns the discrete logarithm $|b_i - c_j|$ (resp. $|b_j - c_i|$).

A heuristic analysis (cf. [Gal12, §14.5]) suggests that for $k \approx \log(W)/2$, $\mu \approx \sqrt{W}/2$, $d \approx \log(W)/\sqrt{W}$, the time cost of this algorithm is $O(\sqrt{W})$ group operations, while the memory cost is negligible.

The objective is now for you to implement this algorithm to search for logarithms in $[\![0, 2^{64} - 1]\!]$ in the subgroup $\mathbb{G} < \mathbb{F}^{\times}_{2^{115}-85}$ of prime order $989008925435205262577237396041921 \approx 2^{109.6}$.

Preparatory work

The file https://membres-ljk.imag.fr/Pierre.Karpman/mul11585.h implements the group law of $\mathbb{F}^{\times}_{2^{115}-85}$, where elements are represented as integers thanks to the union type:

```
typedef union
{
        unsigned __int128 s;
        uint64_t t[2];
} num128;
```

A variable `num128 x` can be accessed either as an unsigned 128-bit integer (which isn't exactly a standard type) as `x.s` or as the two quadwords `x.t[0]`, `x.t[1]` it is made of (typically in little endian).

Question 1

What would be the cost of a generic discrete logarithm computation in the full group $\mathbb{G}$ (i.e. using an algorithm that does not exploit the group structure)? Would this be feasible "in reasonable time" on a personal computer?

Question 2

Write a function `num128 gexp(uint64_t x)` that implements the exponentiation map $[\![0, 2^{64} - 1]\!] \to \mathbb{G}$, $x \mapsto g^x$ where g, represented by the integer 4398046511104, is a generator of $\mathbb{G}$.

You may test your function on the few following values:

— g^{257} = `0x42F953471EDC00840EE23EECF13E4`

— $g^{112123123412345}$ = 0x21F33CAEB45F4D8BC716B91D838CC

— $g^{18014398509482143}$ = 0x7A2A1DEC09D0325357DAACBF4868F

Question 3 (bonus)

Explain how the function `mul11585` works.

Implementing kangaroos

Question 4

Propose an explicit parameterisation and a instantiation strategy of the kangaroo method to solve the stated discrete logarithm problem. That is you must specifiy suitable values for k, μ, d, W and how to pick the exponents $e_{1,\ldots,k}$, the sets $\mathcal{S}_{1,\ldots,k}$ and $\mathcal{D}$.

Question 5

Write a function `num128 dlog64(num128 target)` that solves the stated discrete logarithm problem using the kangaroo method. Use it to compute the discrete logarithm of the element represented by 0x71AC72AF7B138B6263BF2908A7B09.

Question 6

Analyse experimentally the behaviour of your implementation. How does it compare with the heuristic?

Question 7

Tweak some of the parameters of the algorithm (e.g. k, the position of the starting point, etc.) and analyse the impact this has on the experimental running time.

References

[Gal12] Steven D. Galbraith, *Mathematics of Public Key Cryptography*, Cambridge University Press, 2012, Available at https://www.math.auckland.ac.nz/~sgal018/crypto-book/crypto-book.html.