PAWS 2025: MATHEMATICAL CRYPTOGRAPHY PROBLEM SET 2

GIACOMO BORIN, JOLIJN COTTAAR, ELI ORVIS, GABRIELLE SCULLARD

The goal for Problem Set 2 is to practice what we have been learning about attacks on the Discrete Logarithm Problem. The exercises are organized into beginner, intermediate, and advanced levels. If this is your first time thinking about cryptanalysis, the beginner and intermediate problems might be a good place to start. These exercises are meant for your enjoyment and learning, so make sure to work on the problems that interest you the most!

(1) (Beginner) The order of 2 in $\mathbb{F}_{71}^*$ is 35. Charlie uses the subgroup generated by $g = 2$, his public key is $g_c = 29$. Use the baby-step giant-step algorithm to compute an integer c such that $g_c \equiv g^c \mod 71$.

(2) (Intermediate) We have seen in Remark 2.12 (of the lecture notes) that one may reduce the size of the required memory at the cost of increasing the overall runtime of the algorithm. In this exercise, the goal is to achieve the opposite: decreasing the runtime at the cost of increasing the required memory.
  (a) We have shown that Algorithm 1 (of the lecture notes) requires $O(\sqrt{q})$ multiplications. More concretely, show that the average runtime is given by $T = 4/2\sqrt{q}$ (if the exponent a is chosen uniformly at random).

  Now consider a variant of Algorithm 1, where the baby steps and giant steps are computed in parallel, and all values g_i, A_i for $0 \leq i \leq n \leq m$ are stored until the match is found for some n
  (b) What is the average runtime of this variant of Algorithm 1? What is the required memory? Hint: You can use (or prove if you are familiar with probability theory) that for two integers i, j uniformly chosen at random from $\{0, .., m\}$, the expected value of $\max(i, j) \approx 2/3m$.

(3) (sage, Intermediate) Implement the baby-step giant-step algorithm and use it solve the DLP instances from Exercise 5 of the first exercise set (copied again here). How does the running time compare to the *log* function in SageMath? Which algorithm is used in SageMath to solve the DLP?

  In all of these, the public parameters are a prime $p = 2q + 1$, and the element $g = 4 \in \mathbb{F}_p^*$ with order q:
  (a) $q = 4294967681 \approx 2^{32}$,
  $A = 5104411285$, $B = 7620748646$.
  (b) $q = 18446744073709552109 \approx 2^{64}$,
  $A = 17485644247020728566, B = 17485644247020728566$.
  (c) $q = 340282366920938463463374607431768219863 \approx 2^{128}$,
  $A = 15855669586157245378211095347605706305$,
  $B = 64379118553030588585874013496452067220 5$

(4) (Beginner) Use Pollard's rho algorithm to compute c such that $2^c \equiv 29 \pmod{71}$ i.e. use Pollard's rho algorithm to do Exercise 1 in this Problem Set. What are T and L? How does the value of $T + L$ compare to the expected value given in Theorem 2.16?

(5) (Beginner) Explain why $f(x) = x^2$ is a bad (inefficient) choice of function for Pollard's rho algorithm (say, with initial value $x_0 = g \cdot A$, where the order of g is odd).

(6) (Intermediate) *Pollard's ρ for Integer Factorization.* You already know that Pollard's ρ algorithm can be used to solve the *discrete logarithm problem* by exploiting cycle detections. The same idea can be adapted to *factor composite integers*, i.e. find prime numbers $p_1, ..., p_r$ such that $n = p_1 \cdots p_r$.

Let n be a composite integer. Consider the sequence

$$x_{k+1} \equiv \pi(x_k) := x^2 + 1 \pmod{n}.$$

Because there are only n residues, the sequence eventually repeats. If a prime $p \mid n$ causes two values to collide *modulo* p before they collide modulo n, then

$$\gcd\big(|x_i - x_j|, n\big)$$

can reveal a non-trivial factor of n.

(a) (Beginner) Show that if $x_i \equiv x_j \pmod{p}$, then $x_{i+k} \equiv x_{j+k} \pmod{p}$ for all $k \geq 0$.

(b) (sage, Intermediate) Implement Pollard's ρ method of factorization in Sagemath and use it to factor the following integers:
- $n_1 = 1007$;
- $n_2 = 8051$;
- $n_3 = 10403$;
- $n_4 = 5544195998547562675200$;
- $n_5 = 6263601980749376967475 2$;
- $n_6 = 19783$;
- $n_7 = 1631011$.

(Yes, they are ordered from easy to hard :), do the next one if you want to understand why)

(c) (Advanced) Use Theorem 2.16 to estimate the expected number of steps needed to factor n, assuming π to be a random permutation. Note: it may be helpful to use the Theorem on a function different from π, since you are looking for a collision modulo one of the prime factors of n.

(d) (Advanced) Explain why Pollard's ρ is particularly effective when n has many small prime factors, even if n itself is large.

(7) (sage, Intermediate) Implement the Pollard rho algorithm in Sagemath and use it solve the DLP instances from Exercise 3. Compare the run times with your baby-step giant-step implementation.

(8) (Beginner/Intermediate) Use index calculus to compute an integer c such that $2^c \equiv 29 \pmod{71}$ (i.e., use index calculus to solve #1 on this problem set). Use factor base $\mathcal{P}_B = \{2, 3, 5\}$ and the sequence of integers $e = 5, 13, 32, 19$.

(9) (Intermediate) When looking at Example 2.19 from the lecture notes, can you also solve for x_2, x_3 and x_5? Explain your observations.

(10) (sage, Advanced) Try to implement the Index Calculus algorithm (Algorithm 3 from the notes) in Sagemath. Here some hints:
- `list(primes(B))` gives you a list of all primes up to B;
- You can initialize the matrix to then the relations having b columns and $b + 1$ rows.
- You do not need to factor completely, just divide out the primes in your factor base, then if you get 1 it means the number was completely factored.
- You can use `A.solve_right(b)` to solve the linear system $Ax = b$ in $\mathbb{F}_q$.

(11) (Intermediate) Let T_B be the expected number of trials of random integers modulo p until one is B-smooth, b be the number of primes up to B, r is the (prime) order of g and $M(t)$ be the number of bit operations required to multiply two t-bit integers.

Then, the expected running time of the Index Calculus algorithm (using naive trial division and specialized methods for the **sparse** linear system) is:

$$O(b^2 T_B M(\log(p)) + b^{2+o(1)} M(\log(r))) \text{ for } p \to \infty.$$

Let $L_p(c, 1/2)$ be the subexponential function[1] defined as

$$L_p(c, 1/2) = e^{c\sqrt{\log(p)\log(\log(p))}}.$$

Assuming that $B = L_p(c, 1/2)$ for some constant $c > 0$ and $T_B = L_p(1/2, 1/(2c) + o(1))$, show that the optimal value for c is $1/2$. Then estimate the asymptotic complexity of the Index Calculus algorithm.

(12) (sage, Advanced) Use your implementation of the Index Calculus algorithm to run some experiments and find an optimal value for B for $p = 2 \cdot 386545163 + 1$ and $g = 4$. Does your experimental optimal value for B agree with the asymptotic value you found in the previous exercise? How does it change? Try to use larger primes and find the optimal $B = L_p(1/2, c)$ for your implementation experimentally.

[1]if you want to read more see Lemma 15.1.6 from MPKC.