Project 1
Exponentiation in a group

KG

January 9, 2024

1 Introduction

Consider a group G, written multiplicatively with identity $1 \in G$, let $x \in G$ and let $a \in \mathbb{Z}$.

We shall consider two finite groups (p is prime greater than 4 in both cases):

- $\mathbb{F}_p^*$, the non-zero elements of the finite field with p elements, each coset represented by an integer in $\{1, 2, \ldots, p-1\}$.
- $E(\mathbb{F}_p)$, the $\mathbb{F}_p$-rational points on an elliptic curve defined over a field $\mathbb{F}_p$. There are many representation choices. (Note that elliptic curve groups are usually written additively.)

The first group is always cyclic, while the second is sometimes cyclic.

For $a > 0$, we define x^a to be $xx\cdots x$, with a terms. For $a < 0$, we define $x^a = (x^{-1})^{-a}$. For $a = 0$, we define $x^a = 1$.

The definition of exponentiation gives us a trivial algorithm for exponentiation, namely multiply x with itself $a-1$ times (for $a > 0$).

It is sometimes important to be able to compute x^a quickly, and this is the problem we will consider in this exercise: given G, x and a, how do we compute x^a quickly.

2 Tasks

You will write a report using LaTeX. The report should include an explanation of what you have done, a description of your implementation, the theoretical analysis, any experimental results with explanations and a code listing. You should probably use the LaTeX listings package.

1. Warm-up: Modular arithmetic

a. Implement Implement arithmetic for the group $\mathbb{F}_p^*$, including finding inverses (Extended Euclidian algorithm). Implement unit tests.

Note: You may freely reuse any operations for modular arithmetic provided by your programming language (possibly augmented by suitable packages/libraries), except for modular division, inversion and the Extended Euclidian algorithm.

2. Exponentiation

a. Implement Implement the naive exponentiation algorithm based on the definition. Implement a square-and-multiply algorithm from the lectures for the group $\mathbb{F}_p^*$. Also implement unit tests.

b. Analyse Explain the theoretical cost of the two exponentiation algorithms in terms of the size of the prime p and the length of the exponent a.

c. Run Find (or generate) a suitable sequence of primes and use them to time the two implementations of multiplication. For this timing test, let a be non-negative random numbers of the same length as p.

Generate a plot showing the time cost of the two implementations as a function of the size of the prime. Compare the results with the theoretical analysis. Explain any discrepancies.

3. Optional: Advanced methods In this problem, we shall assume that $0 \leq a < p$.

There are faster exponentiation algorithms, and in particular for special cases: Sometimes, we will need to compute g^{a_i} for a fixed $g \in G$ and l (distinct) exponents $a_1, a_2, \ldots, a_l \in \mathbb{Z}$. One tool for speeding up such computations could be precomputations, for instance pre-computing g^{2^i} for many different i. There are other, probably more sensible tricks, such as window methods etc.

Consulting Gordon [4] as well as Menezes *et al.* [5] may be useful.

a. Implement Implement one or more suitable advanced exponentiation algorithms, possibly using precomputation. Also implement unit tests.

b. Analyse Explain the theoretical cost of your chosen algorithms (both the precomputation step and the exponentiation itself) in terms of cost per exponentiation, and compare with the analysis of the square-and-multiply algorithm from Task 2.

c. Run Find (or generate) a suitable sequence of primes together with fixed group elements g for each prime (choose either a generator or a random element), and use them to time the precomputation step and a suitable number of exponentiations (again using random exponents).

Compare the results with the theoretical analysis from Task 2b and with corresponding timings from Task 2. Explain any discrepancies.

4. Optional: Elliptic curves We denote the set of $\mathbb{F}_p$-rational points on an *elliptic curve* $Y^2 = X^3 + AX + B$ over a prime finite field $\mathbb{F}_p$ by $E(\mathbb{F}_p)$. This set forms a group under a particular geometric operation on points.

See Galbraith [1] and Gjøsteen [3] (maybe also [2]).

Note that we usually write the elliptic curve group operation additively (because it is a commutative group), and say addition/doubling about the group operation instead of multiplication/squaring, as well as point multiplication instead of exponentiation. We do not care about this in the next few sentences.

a. Arithmetic Implement arithmetic for the group $E(\mathbb{F}_p)$. Also implement unit tests.

b. Exponentiation Reimplement the fast exponentiation algorithms from Task 2a and 3a for the group $E(\mathbb{F}_p)$. Also implement unit tests.

c. Run Find a suitable sequence of elliptic curves over finite prime fields and use them to time the implementation of the exponentiation algorithms.

Compare the results with the theoretical analysis from Task 2b. Explain any discrepancies.

References

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

[2] Kristian Gjøsteen. Lecture notes in TMA4160 Cryptography. `https://wiki.math.ntnu.no/tma4160/notes`, 2019.

[3] Kristian Gjøsteen. *Practical Mathematical Cryptography*. Chapman and Hall/CRC, 2022.

[4] Daniel M. Gordon. A survey of fast exponentiation methods. *Journal of Algorithms*, 27(1):129–146, 1998.

[5] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996. `https://cacr.uwaterloo.ca/hac/`.