

# 18.783 Elliptic Curves

## Lecture 3

Andrew Sutherland

September 11, 2025

## Representing finite fields

For  $\mathbb{F}_p \simeq \mathbb{Z}/p\mathbb{Z}$  we use integers in  $[0, p - 1]$  denoting elements of  $\mathbb{Z}/p\mathbb{Z}$ .

For  $\mathbb{F}_q \simeq \mathbb{F}_p^d \simeq \mathbb{F}_p[x]/(x^d)$  we use vectors in  $\mathbb{F}_p^d$  denoting elements of  $\mathbb{F}_p[x]/(x^d)$ , which can view as elements of  $\mathbb{F}_p[x]/(f)$  for some irreducible  $f \in \mathbb{F}_p[x]$  of degree  $d$ . It does not matter which  $f$  we pick, but some choices are better than others.

This reduces all computation in finite fields to integer and polynomial arithmetic.

We should note that there are other choices. If  $\mathbb{F}_q^\times = \langle r \rangle$  (so  $r$  is a **primitive root**), we could use 0 to denote 0 and  $e \in [1, q - 1]$  to denote  $r^e$ .

# Integer arithmetic

Complexity of ring operations on  $n$ -bit integers:

---

addition/subtraction	$O(n)$
multiplication (FFT)	$O(n \log n)$ 🏆

---

To multiply polynomials in  $\mathbb{F}_p[x]$  we use **Kronecker substitution**.

Let  $\hat{f} \in \mathbb{Z}[x]$  denote the **lift** of  $f \in \mathbb{F}_p[x]$  to  $\mathbb{Z}[x]$ . We compute  $h = fg \in \mathbb{F}_p[x]$  via

$$\hat{h}(2^m) = \hat{f}(2^m)\hat{g}(2^m)$$

with  $m \geq 2 \lg p + \lg(d + 1)$ , where  $d := \deg f$ . The  $k$ th coefficient of  $h$  can be obtained by extracting the  $k$ th block of  $m$  bits from  $\hat{h}(2^m)$  and reducing it modulo  $p$ .

All ring operations in  $\mathbb{F}_p[x]$  can thus be reduced to ring operations in  $\mathbb{Z}$ , provided we know how to reduce integers modulo  $p$ .

## Euclidean division

For positive integers  $a, b$  we want to compute the unique  $q, r \geq 0$  for which

$$a = bq + r \quad (0 \leq r < b),$$

that is,  $q = \lfloor a/b \rfloor$  and  $r = a \bmod b$ . Recall Newton's method to find a root of  $f(x)$ :

$$x_{i+1} := x_i - \frac{f(x_i)}{f'(x_i)}.$$

To compute  $c \approx 1/b$ , we apply this to  $f(x) = 1/x - b$ , using the Newton iteration

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} = x_i - \frac{\frac{1}{x_i} - b}{-\frac{1}{x_i^2}} = 2x_i - bx_i^2.$$

We can then compute  $q = \lfloor ca \rfloor$  and  $r = a - bq$ .

## Euclidean division

As an example, let us approximate  $1/b = 1/123456789$  working in base 10 (in an actual implementation would use base 2, or base  $2^w$ , where  $w$  is the word size).

$$x_0 = 1 \times 10^{-8}$$

$$\begin{aligned}x_1 &= 2(1 \times 10^{-8}) - (1.2 \times 10^8)(1 \times 10^{-8})^2 \\ &= 0.80 \times 10^{-8}\end{aligned}$$

$$\begin{aligned}x_2 &= 2(0.80 \times 10^{-8}) - (1.234 \times 10^8)(0.80 \times 10^{-8})^2 \\ &= 0.8102 \times 10^{-8}\end{aligned}$$

$$\begin{aligned}x_3 &= 2(0.8102 \times 10^{-8}) - (1.2345678 \times 10^8)(0.8102 \times 10^{-8})^2 \\ &= 0.81000002 \times 10^{-8}.\end{aligned}$$

We double the precision we are using at each step, and each  $x_i$  is correct up to an error in its last decimal place. The value  $x_3$  suffices to correctly compute  $\lfloor a/b \rfloor$  for  $a \leq 10^{15}$ .

## Euclidean division

There is an analogous algorithm for Euclidean division in  $\mathbb{F}_p[x]$ .

Given  $a, b \in \mathbb{F}_p[x]$  with  $b$  monic we can compute the unique  $q, r \in \mathbb{F}_p[x]$  for which

$$a = bq + r \quad (\deg r < \deg b).$$

See the lecture notes for details. In both cases if the divisor  $b$  is fixed we can save time by precomputing  $c \approx 1/b$  (as on Problem Set 1).

### Theorem

*Let  $q = p^e$  be a prime power and assume  $\log e = O(\log p)$  or  $p = O(1)$ .*

*The time to multiply two elements in  $\mathbb{F}_q$  is  $O(n \log n)$ , where  $n = \log q$ .*

Under a widely believed conjecture we know that multiplication in  $\mathbb{F}_q$  takes time  $O(n \log n)$  (but not necessarily  $O(M(n))$ ), without any assumptions about  $p$  and  $d$ .

## Inverting elements of a finite field

Given integers  $a > b > 0$  the (extended) Euclidean algorithm computes  $s, t \in \mathbb{Z}$  with

$$\gcd(a, b) = as + bt \quad (|s| \leq b/\gcd(a, b), |t| \leq a/\gcd(a, b))$$

If  $a = p$  is prime, then  $ps + bt = 1$  and  $t \equiv b^{-1} \pmod{p}$  with  $t \in [0, p - 1]$ .

The Euclidean algorithm works in any Euclidean ring, including  $\mathbb{F}_p[x]$ .

But note that  $\mathbb{F}_p[x]$  has a larger unit group than  $\mathbb{Z}$  and  $\gcd(a, b)$  is defined only up to units. More formally,  $\gcd(a, b) = (a, b) = (c)$  is a principal ideal. In  $\mathbb{Z}$  there is a unique positive choice of  $c$ , while in  $\mathbb{F}_p[x]$  there is a unique monic choice of  $c$ .

The [fast Euclidean algorithm](#) (see lecture notes) yields the following theorem.

### Theorem

*Let  $q = p^d$  be a prime power and assume  $\log d = O(\log p)$  or  $p = O(1)$ .*

*The time to invert an element of  $\mathbb{F}_q^\times$  is  $O(M(n) \log n) = O(n \log^2 n)$ , where  $n = \log q$ .*

# Exponentiation (aka scalar multiplication for multiplicative groups)

Given a group element  $g$  and a positive integer  $a$  we want to compute  $g^a = gg \cdots g$  (or if we write the group operation additively,  $ag = g + g + \cdots + g$ ).

We can achieve this using a “square-and-multiply” (or “double-and-add”) algorithm:

1. Let  $a = \sum_{i=0}^n 2^i a_i$  and initialize  $h$  to  $g$ .
2. For  $i$  from  $n - 1$  down to 0:
  - a. Replace  $h$  with  $h^2$
  - b. If  $a_i = 1$  then replace  $h$  with  $hg$ .

At the end of the  $i$ th loop we have  $h = g^b$  with  $b = \sum_{j=0}^{n-i} 2^j a_{i+j}$ .

This allows us to compute  $g^a$  using at most  $2n = O(n)$  group operations.

The leading constant 2 can be improved (see Problem Set 2).

For  $g \in \mathbb{F}_q^\times$  and  $a \leq q - 1$  the time to compute  $g^a$  is  $O(nM(n)) = O(n^2 \log n)$ .

We can reduce  $a$  modulo  $q - 1$ , and we note that we can compute  $g^{-1} = g^{q-2}$ .

## Root-finding over finite fields

Given  $f \in \mathbb{F}_q[x]$  we wish to compute its  $\mathbb{F}_q$ -rational roots, the set  $\{a \in \mathbb{F}_q : f(a) = 0\}$ .

We can determine the multiplicity of a root  $a$  by evaluating derivatives of  $f$  at  $a$ , since  $(x - a)^j$  divides  $f(x)$  if and only if  $f^{(i)}(a) = 0$  for  $0 \leq i < j$ .

An  $\overline{\mathbb{F}}_q$ -root of  $f$  lies in  $\mathbb{F}_q$  if and only if it is also a root of  $x^q - x$ , thus the  $\mathbb{F}_q$ -rational roots of  $f$  are precisely the roots of  $g(x) := \gcd(f, x^q - x)$ , all of which are distinct.

When  $q$  is larger than  $d := \deg f$ , we do not want to compute  $\gcd(f(x), x^q - x)$  using the Euclidean algorithm; the cost would be superlinear in  $q$  (exponential in  $n = \log q$ ).

Instead we compute  $h(x) = x^q \bmod f$  by exponentiating  $x$  by  $q$  in the ring  $\mathbb{F}_q[x]/(f)$  using binary exponentiation, and then compute  $g(x) := \gcd(f(x), h(x) - x)$ . The polynomial  $g$  that splits into distinct linear factors, one for each  $\mathbb{F}_q$ -rational root of  $f$  (the degree of  $g$  tells us the number of roots).

## Randomized root-finding

Having computed  $g(x) = \gcd(f(x), x^q - x) = (x - a_1) \cdots (x - a_r)$  as a product of monic linear factors whose roots are the  $\mathbb{F}_q$ -rational roots of  $f$ , we already know how many distinct  $\mathbb{F}_q$ -rational roots  $f$  has:  $\deg g$ .

We can use the same approach to compute the number of distinct  $\mathbb{F}_{q^n}$ -rational roots  $f$  has for  $n = 1, 2, \dots, \deg f$ , and by computing their multiplicities we can determine the degrees of all the irreducible factors of  $f \in \mathbb{F}_q[x]$ .

But no polynomial-time algorithm is known for computing the actual roots  $a_1, \dots, a_r$  when  $r > 1$ . We need to use randomization to do this efficiently.. Assume  $q$  is odd.

Rabin: Pick a uniform random  $\delta \in \mathbb{F}_q$  and compute  $h(x) = \gcd(g(x), (x + \delta)^2 + 1)$ .

With probability  $\frac{q-1}{2q}$ , the polynomial  $h$  will be a non-trivial factor of  $g$  (see proof). We can then apply this recursively to  $h$  or  $g/h$  (or both).

## Summary

The table below summarizes the bit-complexity of the various arithmetic operations we have considered, both in the integer ring  $\mathbb{Z}$  and in a finite field  $\mathbb{F}_q$  of cardinality  $q = p^e$ , with  $n$  denoting the bit-size of the inputs and  $M(n)$  the time to multiply in  $\mathbb{F}_q$ .

Assuming  $\log e = O(\log p)$  (or a form of Linnik's conjecture),  $M(n) = O(n \log n)$ .

	integers $\mathbb{Z}$	finite field $\mathbb{F}_q$
addition/subtraction	$O(n)$	$O(n)$
multiplication	$O(n \log n)$	$M(n)$
Euclidean division (reduction)	$O(n \log n)$	$O(M(n))$
extended gcd (inversion)	$O(n \log^2 n)$	$O(M(n) \log n)$
exponentiation		$O(nM(n))$
square-roots (probabilistic)		$O(nM(n))$
root-finding (probabilistic)		$O(M(d(n + \log d))(n + \log d))$
factoring (probabilistic)		$O(M(d(n + \log d))d(n + \log d))$
irreducibility testing		$O(M(d(n + \log d))d(n + \log d))$

MIT OpenCourseWare

<https://ocw.mit.edu>

18.783 / 18.7831 Elliptic Curves

Fall 2025

For information about citing these materials or our Terms of Use, visit: <https://ocw.mit.edu/terms>.