11 Index calculus, smooth numbers, and factoring integers

Having considered a variety of generic algorithms for the discrete logarithm problem, we now want to consider a non-generic algorithm based on a technique known as *index calculus*.[1] This algorithm depends critically on the distribution of *smooth numbers* (integers whose prime factors are all small), and this leads naturally to a discussion of two methods for factoring integers: the Pollard $p-1$ method and the elliptic curve method (ECM).

11.1 Index calculus

Index calculus is a method for computing discrete logarithms in the multiplicative group of a finite field. This might not seem directly relevant to the elliptic curve discrete logarithm problem, but as we shall see when we discuss pairing-based cryptography, these two problems are not unrelated. Moreover, the same approach can be applied to elliptic curves over non-prime finite fields, as well as abelian varieties of higher dimension [7, 8, 9].[2]

We will restrict our attention to the simplest case, a finite field $\mathbb{F}_p$ of prime order. Let us identify $\mathbb{F}_p \simeq \mathbb{Z}/p\mathbb{Z}$ with our usual choice of representatives for $\mathbb{Z}/p\mathbb{Z}$, the set of integers in the interval $[0, p-1]$. This seemingly innocuous statement is precisely what will allow us to "cheat", that is, to do something that a generic algorithm cannot. It lets us "lift" elements of $\mathbb{F}_p \simeq \mathbb{Z}/p\mathbb{Z}$ to the ring $\mathbb{Z}$ where we may consider their prime factorizations, something that makes no sense in a multiplicative group (since every element is invertible) and is not available to a generic algorithm. The reduction map $\mathbb{Z} \to \mathbb{Z}/p\mathbb{Z}$ allows us to use these prime factorizations relations to obtain relations between the discrete logarithms of different elements of $\mathbb{F}_p^\times$.

Let us fix a generator α for $\mathbb{F}_p^\times$, and let $\beta \in \mathbb{F}_p^\times$ be the element whose discrete logarithm we wish to compute. For any integer e, we may consider the prime factorization of the integer $\alpha^e\beta^{-1} \in [1, N] \subseteq \mathbb{Z}$. When $e = \log_\alpha \beta$ this prime factorization will be trivial, but in general we will have

$$\prod p_i^{e_i} = \alpha^e \beta^{-1},$$

where the p_i vary over primes the exponents e_i are nonnegative integers. Multiplying both sides by β and taking discrete logarithms with respect to α yields

$$\sum e_i \log_\alpha p_i + \log_\alpha \beta = e,$$

which determines $\log_\alpha \beta$ as a linear expression in the discrete logarithms $\log_\alpha p_i$ (note that we are viewing the p_i both as primes in $\mathbb{Z}$ and elements of $\mathbb{F}_p$). As it stands this doesn't immediately help us, since we don't know the values of the various $\log_\alpha p_i$. However, if we repeat this procedure for many different values of e, we may obtain a system of linear equations that we can then solve for $\log_\alpha \beta$.

[1] If α is a generator for $\mathbb{F}_p^\times$ then the discrete logarithm of $\beta \in \mathbb{F}_p^\times$ with respect to α is also called the *index* of β (with respect to α), whence the term *index calculus*.

[2] The two are related: if E is an elliptic curve over a finite field $\mathbb{F}_{q^n}$ for some prime-power q, there is an associated abelian variety of dimension n over $\mathbb{F}_q$ known as the *Weil restriction* of E.

In order to make this feasible, we need to restrict the set of primes p_i that we are going to work with. We thus pick a *smoothness bound*, say B, and define the *factor base*

$$P_B = \{p : p \le B \text{ is prime}\} = \{p_1, p_2, \dots, p_b\},$$

where $b = \pi(B)$ is the number of primes up to B (of which there are approximately $B/\log B$, by the Prime Number Theorem). Not all choices of e will yield an integer $\alpha^e\beta^{-1} \in [1, N] \subseteq \mathbb{Z}$ that we can write as a product of primes in our factor base P_B, in fact most won't. But some choices will work, and for those that do we obtain a linear equation of the form

$$e_1 \log_\alpha p_1 + e_2 \log_\alpha p_2 + \cdots + e_b \log_\alpha p_b = e,$$

(many of the e_i may be zero). We don't yet know any of the discrete logarithms on the LHS, but we can view

$$e_1x_1 + e_2x_2 + \cdots + e_bx_b + x_{b+1} = e$$

as a linear equation in $b+1$ variables $x_1, x_2, \dots, x_{b+1}$ over the ring $\mathbb{Z}/N\mathbb{Z}$. This equation has a solution, namely, $x_i = \log_\alpha p_i$, for $1 \le i \le b$, and $x_{b+1} = \log_\alpha \beta$. If we collect $b+1$ such equations by choosing random values of e and discarding those for which $\alpha^e\beta^{-1}$ is not B-smooth, we may be able to solve the resulting linear system for the x_i, and in particular for x_{b+1}, which is the discrete logarithm we wish to compute.

This system will typically be under-determined; indeed, some variables x_i may not appear in any equation. But it is quite likely that the value of x_{b+1}, which is guaranteed to be present in every equation, will be uniquely determined. We will not attempt to prove this (to give a rigorous proof one really needs more than $b+1$ equations, say, on the order of $b \log b$), but it is empirically true.[3]

This suggests the following algorithm to compute $\log_\alpha \beta$.

Algorithm 11.1 (Index calculus in a prime field $\mathbb{F}_p$)**.**

1. Pick a smoothness bound B and construct the factor base $P_B = \{p_1, \dots, p_b\}$.
2. Generate $b+1$ random relations $R_i = (e_{i,1}, e_{i,2}, \dots, e_{i,b}, 1, e_i)$ by picking $e \in [1, N]$ at random and and attempting to factor $\alpha^e\beta^{-1} \in [1, N]$ over the factor base P_B. Each successful factorization yields a relation R_i with $e_i = e$ and $\alpha^{e_i}\beta^{-1} = \prod p_j^{e_{i,j}}$.
3. Attempt to solve the system defined by the relations $R_1, \dots, R_{b+1}$ for $x_{b+1} \in \mathbb{Z}/N\mathbb{Z}$ using linear algebra (e.g., row reduce the corresponding matrix).
4. If $x_{b+1} = \log_\alpha \beta$ is uniquely determined, return this value, otherwise go to step 2.

It remains to determine the choice of B in step 1, but we first make the following remarks.

Remark 11.2. It is not actually necessary to start over from scratch when x_{b+1} is not uniquely determined, typically adding just a few more relations will be enough.

Remark 11.3. The relations $R_1, \dots, R_{b+1}$ will be *sparse* (have few nonzero entries). The linear algebra step can be accelerated by using algorithms that take advantage of this fact.

[3] When considering potential attacks on a cryptographic system, one should err on the side of generosity when it comes to heuristic assumptions that help the attack.

Remark 11.4. While solving the system $R_1, \ldots, R_{b+1}$ we are likely to encounter zero divisors in the ring $\mathbb{Z}/N\mathbb{Z}$ (for example, 2 is always a zero divisor, since $N = p - 1$ is even). Whenever this happens we can use a gcd computation to obtain a non-trivial factorization $N = N_1 N_2$ with N_1 and N_2 relatively prime. We then proceed to work in $\mathbb{Z}/N_1\mathbb{Z} \times \mathbb{Z}/N_2\mathbb{Z}$, using the CRT to recover the value of x_{b+1} in $\mathbb{Z}/N\mathbb{Z}$ (recurse as necessary).

Remark 11.5. Solving the system of relations will generally determine the value of not only $x_{b+1} = \log_\alpha \beta$, but also of many of the $x_i = \log_\alpha p_i$ for $p_i \in P_B$, which do note depend on β. If we are computing discrete logarithms for many different β with respect to the same base α, after the first computation the number of relations we need is just one more than the number of $x_i = \log_\alpha p_i$ that have yet to be determined. If we are computing discrete logarithms for $\Omega(b)$ values of β, we expect to compute just $O(1)$ relations per discrete logarithm, on average.

Let us now consider the smoothness bound B. We first note that α^e, and therefore $\alpha^e\beta^{-1}$, is uniformly distributed over $\mathbb{F}_p^\times$, which we have identified with the set of integers in $[1, N]$. An integer whose prime factors are all bounded by B is said to be *B-smooth*. A large value of B will make it more likely that $\alpha^e\beta^{-1}$ is B-smooth, but it also makes it more difficult to determine whether this is in fact the case, since we must verify that all the prime factors of $\alpha^e\beta^{-1}$ are bounded by B. To determine the optimal value of B, we want to balance the cost of smoothness testing against the number of smoothness tests we expect to need in order to get $b + 1$ relations (note that b depends on B). Let us suppose for the moment that the cost of the linear algebra step is negligible by comparison (which turns out to be the case, at least in terms of time complexity). In order to choose B, we need to know the probability that a random integer in the interval $[1, N]$ is B-smooth.

11.2 The Canfield-Erdös-Pomerance Theorem

For any positive real numbers x and y, let $\psi(x, y)$ be the number of y-smooth positive integers bounded by x. The probability that a positive integer $m \le x$ is y-smooth is then approximately $\frac{1}{x}\psi(x, y)$. Now let

$$u = \frac{\log x}{\log y},$$

so that $y = x^{1/u}$. The Canfield-Erdős-Pomerance Theorem [6] states that the bound

$$\frac{1}{x}\psi(x, x^{1/u}) = u^{-u+o(u)}$$

holds uniformly as $u, x \to \infty$, provided that $u < (1 - \epsilon) \log x / \log\log x$ for some $\epsilon > 0$. For more on this result (which we will not prove here) along with many other interesting facts about smooth numbers, we recommend the survey article by Granville [12].

11.3 Optimizing the smoothness bound

Let us assume that generating relations dominates the overall complexity of Algorithm 11.1, and for the moment suppose that we simply use trial-division to attempt to factor $\alpha^e\beta^{-1}$ over P_B (we will see a more efficient method for smoothness testing shortly). Then the expected running time of Algorithm 11.1 is approximately

$$(b + 1) \cdot u^u \cdot b \cdot \mathsf{M}(\log N), \tag{1}$$

where $u = \log N / \log B$. The four factors in (1) correspond, respectively, to:

- the number of relations R_i that we need,
- the expected number of random exponents e needed to obtain a B-smooth integer $m = \alpha^e \beta^{-1} \in [1, N]$,
- the number of trial divisions to test whether m is B-smooth (and factor it over P_B),
- the time for each trial division.

We have $b = \pi(B) \sim B/\log B$, and by ignoring logarithmic factors, we can replace $b+1$ and b by B and drop the $\mathsf{M}(\log N)$ factor. We then wish to choose u to minimize the quantity

$$B^2 u^u = N^{2/u} u^u, \tag{2}$$

where we have used $B^u = N$ to eliminate B. Taking logarithms, it suffices to minimize

$$f(u) = \log(N^{2/u} u^u) = \frac{2}{u} \log N + u \log u,$$

add we thus set

$$f'(u) = -\frac{2}{u^2} \log N + \frac{2}{uN} + \log u + 1 = 0.$$

Ignoring the asymptotically negligible terms 1 and $\frac{2}{uN}$, we would like to pick u so that

$$u^2 \log u \approx 2 \log N.$$

If we let

$$u = 2\sqrt{\log N / \log\log N}, \tag{3}$$

then

$$u^2 \log u = \frac{4 \log N}{\log\log N} \cdot \left(\log 2 + \frac{1}{2}(\log\log N - \log\log\log N) \right) = 2 \log N + o(\log N),$$

as desired. The choice of u in (3) implies that we should use the smoothness bound

$$\begin{aligned} B = N^{1/u} &= \exp\left(\frac{1}{u} \log N \right) \\ &= \exp\left(\frac{1}{2} \sqrt{\log N \log\log N} \right) \\ &= L_N[1/2, 1/2]. \end{aligned}$$

Here we have used the standard asymptotic notation

$$L_N[\alpha, c] = \exp\big((c + o(1)) (\log N)^\alpha (\log\log N)^{1-\alpha} \big).$$

Note that

$$L_N[0, c] = \exp((c + o(1) \log\log N) = (\log N)^{c+o(1)}$$

is polynomial in $\log N$, whereas

$$L_N[1, c] = \exp((c + o(1)) \log N) = N^{c+o(1)}$$

is exponential in $\log N$. For $0 < \alpha < 1$ the bound $L_N[\alpha, c]$ is said to be *subexponential*.

We also have $u^u = \exp(u \log u) = L_N[1/2, 1]$, thus the total expected running time is

$$B^2 u^u = L_N[1/2, 1/2]^2 \cdot L_N[1/2, 1] = L_N[1/2, 2].$$

The cost of the linear algebra step is certainly no worse than $\widetilde{O}(b^3)$. We may bound this by $\widetilde{O}(B^3)$, which in our subexponential notation is $L_N[1/2, 3/2]$. So our assumption that the cost of generating relations dominates the running time is justified. In fact, if done efficiently (and taking advantage of sparseness), the cost of the linear algebra step is closer to $\widetilde{O}(b^2)$. However, in large computations the linear algebra step can become a limiting factor because it is memory intensive and not as easy to parallelize.

Remark 11.6. As noted earlier, if we are computing many (say at least $L_N[1/2, \sqrt{2}/2]$) discrete logarithms with respect to the same base α, we just need $O(1)$ relations per β, on average. In this case we should choose $B = N^{1/u}$ to minimize Bu^u rather than $B^2 u^u$. This yields an average expected running time of $L_N[1/2, \sqrt{2}]$ per β.

A simple version of Algorithm 11.1 using trial-division for smoothness testing is implemented in the Sage worksheet 18.783 Lecture 11: Index Calculus.sws.

11.4 Improvements

Using the elliptic curve factorization method (ECM) which we will discuss in the next section, the cost of testing and factoring B-smooth integers can be made subexponential in B and polynomial in $\log N$. This effective changes $B^2 u^u$ in (2) to Bu^u, and the optimal smoothness bound becomes $B = L_N[1/2, 1/\sqrt{2}]$, yielding a heuristic expected running time of

$$L_N[1/2, \sqrt{2}].$$

There is a batch smoothness testing algorithm due to Bernstein [3] that for a sufficiently large set of integers yields an average time per integer that is actually polynomial in $\log N$, but this doesn't change the complexity in a way that is visible in our $L_N[\alpha, c]$ notation.

Using more advanced techniques, analogous to those used in the *number field sieve* for factoring integers, one can achieve a heuristic expected running time of

$$L_N[1/3, c]$$

for computing discrete logarithms in $\mathbb{F}_p^\times$ (again using an index calculus approach); see [11].

In finite fields of small characteristic $\mathbb{F}_{p^n} \simeq \mathbb{F}_p[x]/(f(x))$, one use the *function field sieve*, where now the factor base consists of low degree polynomials in $\mathbb{F}_p[x]$ that represent elements of $\mathbb{F}_p^n$ when reduced modulo $f(x)$. This also yields an $L_N[1/3, c]$ bound (with an even better value of c), and it is now known that such a bound holds (under heuristic assumptions) for all finite fields [14].

But this is far from the end of the story. In 2013 Antoine Joux announced an index calculus approach for finite fields of small characteristic and composite degree that heuristically achieves an $L_N[1/4 + o(1), c]$ time complexity [13]. Not long thereafter a recursive variant of Joux's approach was used to obtain a heuristically quasi-polynomial-time algorithm, that is, one that runs in time polynomial in $(\log N)^{\log \log N}$, which is $L_N[\epsilon, c]$ for every $\epsilon > 0$; see [2]. This algorithm is constrained to finite fields that can be written in the form $\mathbb{F}_{q^k}$, where q is a prime power and $q \approx k$, but it is quite practical and has produced a flurry of recent records for computing discrete logarithms in finite fields of small characteristic

and composite extension degree. As of January 2014, the record computation for fields of characterstic 2 was over the finite field $\mathbb{F}_{2^{9234}}$; the computation used approximately 400,000 core hours [10]. By contrast the largest discrete logarithm computation over a prime field $\mathbb{F}_p$ for which $p-1$ has a large prime factor (to prevent a Pohlig-Hellman attack) involves a 596-bit prime [5].

11.5 The Pollard $p-1$ method

We now want to look at algorithms for factoring integers that also rely on smooth numbers. In 1974, Pollard introduced a Monte Carlo algorithm for factoring integers [17] that works astonishingly well when the integer $p-1$ is extremely smooth (but in the worst case is no better than trial division). The algorithm takes as input an integer N to be factored and a smoothness bound B.

Algorithm 11.7 (Pollard $p-1$ factorization)**.**
Input: An integer N to be factored and a smoothness bound B.
Output: A proper divisor of N or `failure`.

1. Pick a random integer $a \in [1, N-1]$.
2. If $d = \gcd(a, N)$ is not 1 then return d.
3. Set $b = a$ and for each prime $\ell \le B$:
   a. Set $b = b^{\ell^e} \bmod N$, where $\ell^{e-1} < N \le \ell^e$.
   b. If $d = \gcd(b-1, N)$ is not 1 then return d if $d < N$ or `failure` if $d = N$.
4. Return `failure`

Rather than using a fixed bound B, we could simply let the algorithm keep running through primes ℓ until it either succeeds or fails in step 3b. But in practice one typically uses a very small smoothness bound B and switches to a different algorithm if the $p-1$ method fails. In any case, it is convenient to have B fixed for the purposes of analysis.

Theorem 11.8. *Let p and q be prime divisors of N, and let ℓ_p and ℓ_q be the largest prime divisors of $p-1$ and $q-1$, respectively. If $\ell_p \le B$ and $\ell_p < \ell_q$ then Algorithm 11.7 succeeds with probability at least $1 - \frac{1}{\ell_q}$.*

Proof. If $a \equiv 0 \bmod p$ then the algorithm succeeds in step 2, so we may assume $a \perp p$. When the algorithm reaches $\ell = \ell_p$ in step 3 we have $b = a^m$, where $m = \prod_{\ell \le \ell_p} \ell^e$ is a multiple of $p-1$. By Fermat's little theorem, $b = a^m \equiv 1 \bmod p$ and therefore p divides $b-1$. But ℓ_q does not divide m, so with probability at least $1 - \frac{1}{\ell_q}$ we have $b \not\equiv 1 \bmod q$, in which case $1 < \gcd(b-1, N) < N$ in step 3b and the algorithm succeeds. □

For most values of N, the Algorithm 11.7 will succeed with very high probability if given the smoothness bound $B = \sqrt{N}$. But if N is a prime power, or if the largest prime dividing $p-1$ is the same for every prime factor p of N it will still fail, no matter what value of a is chosen. In the best case, the algorithm can succeed very quickly. As demonstrated in the Sage worksheet 18.783 Lecture 12: Pollard p-1.sws, if $N = p_1 p_2$ where p_1 and p_2 are 512-bit primes, if $p_1 - 1$ happens to be very smooth then Alogorithm 11.7 can factor N within a few seconds, whereas there is essentially no other algorithm known that has any

hope of factoring N in any reasonable amount of time. However, in the worst-case the running time is $O(\pi(B)\mathsf{M}(\log N)\log N)$, and with $B = \sqrt{N}$ the complexity is $O(\sqrt{N}\mathsf{M}(\log N))$ time, which is the same as for trial division.

But rather than focusing on factoring a single integer N, let us consider a slightly different problem. Suppose we have a large set of composite integers (for example, a list of RSA[4]), and our goal is to factor any one of them. How long would this take if we simply applied the $p-1$ method to each integer one-by-one, using a suitably chosen smoothness bound B?

For a given value of B, the expected time for the algorithm to achieve a success is

$$\frac{O(\pi(B)\mathsf{M}(\log N)\log N)}{\Pr[\texttt{success}]}. \tag{4}$$

Let p be a prime factor of N. The algorithm is very likely to succeed if $p-1$ is B-smooth, since it is very unlikely that all the other prime factors q of N have $q-1$ with the exact same largest prime factor as $p-1$. Let us heuristically assume that integers of the form $p-1$ are at least as likely to be smooth as a random integer of similar size. One can argue that they are actually slightly more likely to be smooth since, for example, they must be divisible by 2 and have a 50/50 chance of being divisible by 3.

By the Canfield-Pomerance-Erdős Theorem, the probability that a random integer less than N is B-smooth is $u^{-u+o(u)}$, where $u = \log N/\log B$. If we ignore the $o(u)$ error term and factors that are polynomial in $\log N$ (which will be bounded by $o(u)$ in any case), we may simplify (4) to

$$N^{1/u}u^u. \tag{5}$$

This is minimized (up to asymptotically negligible factors) for $u = \sqrt{2\log N/\log\log N}$, thus we should use the smoothness bound

$$B = N^{1/u} = \exp\left(\left(1/\sqrt{2}+o(1)\right)\sqrt{\log N\log\log N}\right) = L_N[1/2, 1/\sqrt{2}],$$

where the $o(1)$ term incorporates the $o(u)$ error term and the factors polynomial in $\log N$ that we have ignored. We also have $u^u = \exp(u\log u) = L_N[1/2, 1/\sqrt{2}]$, and the total expected running time is therefore

$$N^{1/u}u^u = L_N[1/2, 1/\sqrt{2}]L_N[1/2, 1/\sqrt{2}] = L_N[1/2, \sqrt{2}].$$

Thus even though the $p-1$ method has an exponential worst-case running time, if we apply it to a sequence of random integers we achieve a (heuristically) subexponential running time. But this isn't much help if there is a particular integer N that we need to factor.

11.6 The elliptic curve method for factoring integers (ECM)

Using elliptic curves we can effectively achieve the randomized scenario envisioned while keeping N fixed. The Pollard $p-1$ works in the group $(\mathbb{Z}/N\mathbb{Z})^*$, which, by the Chinese remainder theorem, we can think of as a simultaneous computations in the groups $(\mathbb{Z}/p\mathbb{Z})^\times$ for primes $p|N$; it succeeds when one of these groups has smooth order. If we instead take an elliptic curve $E/\mathbb{Q}$ defined by an integral equation $y^2 = x^3 + Ax + B$ that we can reduce

[4] In fact, many RSA key generation algorithms incorporate specific measures to prevent the type of attack we consider here. In any case, current RSA keys are necessarily large enough (2048 bits) to be quite safe from the $L_N[1/2, \sqrt{2}]$ algorithm considered here.

modulo N, we have an opportunity to factor N whenever $E(\mathbb{F}_p)$ has smooth order, for some prime $p|N$. The key difference is that now we can vary the curve E while keeping N fixed, and we get new groups $E(\mathbb{F}_p)$ each time we change E. This is the basis of the elliptic curve method (ECM), introduced by Hendrik Lenstra [15] in the mid 1980s.

The algorithm is essentially the same as Pollard's $p-1$ method. Rather than exponentiating a random element of $(\mathbb{Z}/N\mathbb{Z})^*$ to a large smooth power and hoping that it becomes the identity modulo some prime p dividing N, we instead multiply a random point on an elliptic curve by a large smooth scalar and hope that it becomes the identity modulo some prime p dividing N. If this doesn't happen we can simply switch to a different curve and try again.

As in Pollard's algorithm, we don't know the primes p dividing N, so we work modulo N and use GCD's to find a factor of N. If P is a point on $E/\mathbb{Q}$ and $mP = (Q_x : Q_y : Q_z)$ is a multiple of P that reduces to 0 modulo a prime p dividing N, then p divides $\gcd(Q_z, N)$. Notice that even though we are working with points on an elliptic curve over $\mathbb{Q}$, we only care about their reductions modulo primes dividing N, so we can keep the coordinates reduced modulo N throughout the algorithm.

In order to get a proper divisor of N we also need $\gcd(Q_z, N) \neq N$. This is very likely to be the case, so long as P is not a torsion point of $E(\mathbb{Q})$; if P is a torsion point it will have the same order modulo every prime divisor of N and we will always have $\gcd(Q_z, N) = N$ whenever the gcd is non-trivial. Given an elliptic curve $E/\mathbb{Q}$, it is generally hard to find non-torsion points in $E(\mathbb{Q})$, in fact there may not be any.[5] Instead we pick integers $x_0, y_0, a \in [1, N-1]$ and let $b = y_0^2 - x_0^3 - ax_0$. This guarantees that $P = (x_0, y_0)$ is a rational point on the elliptic curve $E/\mathbb{Q}$ defined by $y^2 = x^3 + ax + b$. The probability that P is a torsion point is negligible.[6] We now give the algorithm, which takes not only an integer N and a smoothness bound B, but also a bound M on the largest prime factor of N that we seek to find (as discussed below, this is useful for smoothness testing).

Algorithm 11.9 (ECM)**.**
Input: An integer N to be factored, a smoothness bound B, and a prime bound M.
Output: A proper divisor of N or `failure`.

1. Pick random integers $a, x_0, y_0 \in [0, N-1]$ and set $b = y_0^2 - x_0^3 - ax_0$.
2. If $d = \gcd(4a^3 + 27b^2, N)$ is not 1 then return d if $d < N$ or `failure` if $d = N$.
3. Let $Q = P = (x_0 : y_0 : 1)$.
4. For all primes $\ell < B$:
   a. Set $Q = \ell^e Q \bmod N$, where $\ell^{e-1} \leq (\sqrt{M}+1)^2 < \ell^e$.
   b. If $d = \gcd(Q_z, N)$ is not 1 then return d if $d < N$ or `failure` if $d = N$.
5. Return `failure`.

The scalar multiplication in step 4a is performed using projective coordinates, and while it is defined in terms of the group operation in $E(\mathbb{Q})$, we only keep track of the coordinates of Q modulo N; the projective coordinates are integers and there are no inversions involved, so all of the arithmetic can be performed in $\mathbb{Z}/N\mathbb{Z}$.

[5] There are standard parameterizations that are guaranteed to produce a curve $E/\mathbb{Q}$ with a known point $P \in E(\mathbb{Q})$ of infinite order; see [1], for example. Here we just generate random E and P at random.

[6] This follows (for example) from the Lutz–Nagell theorem [18, Theorem 8.7], which implies that if y_0 is nonzero then y_0^2 must divide $4a^3 + 27b^2 = 4a^3 + 27(x_0^3 + ax_0)^2$, which is extremely unlikely.

Theorem 11.10. *Assume $4a^3 + 27b^2$ is not divisible by N, and let P_1 and P_2 be the reductions of P modulo distinct primes p_1 and p_2 dividing N, with $p_1 \le M$. Suppose $|P_1|$ is ℓ_1-smooth and $|P_2|$ is not, for some prime $\ell_1 \le B$. Then Algorithm 11.9 succeeds.*

Proof. When the algorithm reaches step 2b with $\ell = \ell_1$ we must have $Q = mP$, where $m = \prod_{\ell \le \ell_1} \ell^e$ is a multiple of $|P_1|$, since $|P_1|$ is ℓ_1-smooth and $|P_1| \le (\sqrt{p_1}+1)^2 \le (\sqrt{M}+1)^2$. So $Q \equiv 0 \bmod p_1$, but $Q \not\equiv 0 \bmod p_2$, since $|P_2|$ is not ℓ_1-smooth. Therefore Q_z is divisible by p_1 but not p_2 and a proper factor $d = \gcd(Q_z, N)$ of N will be found in step 4b. □

If the algorithm fails, we can simply try again. Heuristically, provided N is not a perfect power and has a prime factor $p \le M$, we will eventually succeed. Factoring perfect powers can be efficiently handled by the algorithm developed in Problem 1 of Problem Set 3. Provided N is not a prime power and has a prime factor $p < M$, Algorithm 11.9 is very likely to succeed whenever it picks a triple (x_0, y_0, a) that yields an elliptic curve whose reduction modulo p has B-smooth order. So the number of times we expect to run the algorithm before we succeed depends on the probability that $\#E(\mathbb{F}_p)$ is B-smooth.

The integer $\#E(\mathbb{F}_p)$ must lie in the Hasse interval $[p - 2\sqrt{p} + 1, p + 2\sqrt{p} + 1]$, which is unfortunately too narrow for us to apply any theorems on the density of B-smooth integers (we cannot even prove that this interval contains any primes, and smooth numbers are much rarer than primes). So to analyze the complexity of Algorithm 11.9 (and to optimize the choice of B), we resort to the heuristic assumption that, at least when $\#E(\mathbb{F}_p)$ lies in the narrower interval $[p + 1 - \sqrt{p} + 1, p + 1 + \sqrt{p}]$, the probability the $\#E(\mathbb{F}_p)$ is B-smooth is comparable to the probability that a random integer in the interval $[p, 2p]$ is B-smooth.[7]

One can prove that the probability that $\#E(\mathbb{F}_p)$ lies in $[p - \sqrt{p} + 1, p + \sqrt{p} + 1]$ is at least $1/2$ (this is implied, asymptotically, by the Sato–Tate theorem), and further that probability that $\#E(\mathbb{F}_p)$ takes on any particular value in this interval is $\Omega(1/(\sqrt{p}\log p))$. These facts are both proved in Lenstra's paper [15], and we will be able to prove them ourselves once we have covered the theory of complex multiplication. This means that we can make our heuristic assumption independent of any facts about elliptic curves, we simply need to assume that a random integer in the interval $[p + 1 - \sqrt{p}, p + 1 + \sqrt{p}]$ has roughly the same probability of being B-smooth as a random integer in the interval $[p, 2p]$.

Under our heuristic assumption, the analysis of the algorithm follows the analysis of the Pollard $p - 1$ method. This algorithm takes $O(\pi(B)(\log M)\mathsf{M}(\log N))$ time per elliptic curve, and if N has a prime factor $p \le M$, it will need to try an average of $O(u^u)$ curves before it finds a factor. As in §11.5, this implies that the optimal value of B is $L_M[1/2, 1/\sqrt{2}]$, and with this value of B the expected time to factor N is $L_M[1/2, \sqrt{2}]\mathsf{M}(\log N)$. In general, we may not know a bound M on the smallest prime factor p of N *a priori*, but if we simply start with a small choice of M and periodically double it, we can achieve a running time of

$$L_p[1/2, \sqrt{2}]\mathsf{M}(\log N),$$

where p is the smallest prime factor of N.

A crucial point is that this running time depends almost entirely on p rather than N, a property that distinguishes ECM from all other factorization algorithms with heuristically subexponential running times. There are factorization algorithms such as the quadratic sieve and the number field sieve that are heuristically faster when all of the prime factors of N are large, but in practice one first uses ECM to look for any relatively small prime factors before resorting to these heavyweight algorithms.

[7] Asymptotically, this is the same as the probability that a random integer in $[1, p]$ is B-smooth.

The fact that the complexity of ECM depends primarily on the size of the smallest prime divisor of N also makes it a very good algorithm for smoothness testing. Testing whether a given integer N is $L_N[1/2, c]$-smooth using ECM takes just

$$\begin{aligned} L_{L_N[1/2,c]}\left[1/2, \sqrt{2}\right] &\approx \exp\left(\sqrt{2\log(\exp(c\sqrt{\log N \log\log N})\log\log(\exp(c\sqrt{\log N \log\log N})}\right) \\ &= \exp\left(\sqrt{2c\sqrt{\log N \log\log N}(1/2\log\log N + o(\log\log N))}\right) \\ &= \exp\left(\sqrt{c}(\log N)^{1/4}(\log\log N)^{3/4}\right) \\ &= L_N\left[1/4 + o(1), \sqrt{c}\right] \end{aligned}$$

expected time, which is faster than any other method known.[8]

11.7 Efficient implementation

Algorithm 11.9 spends essentially all of its time performing elliptic curve scalar multiplications modulo N, so it is worth choosing the elliptic curve representation and the coordinate system to optimize this operation. Edwards curves, which we saw in Lecture 2, are an excellent choice; see [4] for a detailed discussion of how to efficiently implement ECM using Edwards curves. Another popular choice is Montgomery curves [16]. These were originally introduced specifically for the purpose of optimizing the elliptic curve factorization method but are now used in many other applications of elliptic curves, including primality proving and cryptography.

11.8 Montgomery Curves

A *Montgomery curve* is an elliptic curve defined by an equation of the form

$$By^2 = x^3 + Ax^2 + x, \tag{6}$$

where $B \neq 0$ and $A \neq \pm 2$. To convert this to Weierstrass form, let $u = Bx$ and $w = B^2y$. Substituting $x = u/B$ and $y = w/B^2$ in (6) and multiplying by B^3 yields

$$w^2 = u^3 + ABu^2 + B^2u,$$

which is in the form of a general Weierstrass equation. To obtain a short Weierstrass equation, we assume our base field has characteristic different from 3 and complete the cube by letting $v = u + \frac{AB}{3}$. We then obtain

$$\begin{aligned} w^2 &= u^3 + ABu^2 + B^2u \\ w^2 &= \left(v - \frac{AB}{3}\right)^3 + AB\left(v - \frac{AB}{3}\right)^2 + B^2\left(v - \frac{AB}{3}\right) \\ w^2 &= v^3 - ABv^2 + \frac{A^2B^2}{3}v - \frac{A^3B^3}{27} + ABv^2 - \frac{2A^2B^2}{3}v + \frac{A^3B^3}{9} + B^2v - \frac{AB^3}{3} \\ w^2 &= v^3 + \left(B^2 - \frac{A^2B^2}{3}\right)v + \left(\frac{2A^3B^3}{27} - \frac{AB^3}{3}\right). \end{aligned}$$

[8] As noted earlier, for batch smoothness testing, Bernstein's algorithm [3] is faster.

In order to check that (6) actually defines an elliptic curve, we should verify that it is nonsingular. We could do these using the coefficients of the curve in short Weierstrass form, but it is easier to do this directly. We need to determine whether there are any points $(x : y : z)$ on the projective curve $By^2z = x^3 + Ax^2z + xz^2$ at which all three partial derivatives vanish. For any such point we must have

$$\frac{\partial}{\partial x}\colon 3x^2 + 2Axz + z^2 = 0, \qquad \frac{\partial}{\partial y}\colon 2Byz = 0, \qquad \frac{\partial}{\partial z}\colon By^2 = Ax^2 + 2xz = 0.$$

We assume we are working in a field of characteristic not equal to 2 or 3. Suppose that $y \neq 0$. Then the equation for $\frac{\partial}{\partial y}$ gives $z = 0$, and from $\frac{\partial}{\partial x}$, we get $x = 0$. But this is a contradiction, since the equation for $\frac{\partial}{\partial z}$ is not satisfied. On the other hand, if $y = 0$, then $z = -\frac{A}{2}x \neq 0$. We have $3x^2 - A^2x^2 + \frac{A^2}{4}x^2 = 0$, and therefore $3 - \frac{3}{4}A^2 = 0$, since $x \neq 0$. Thus $A^2 = 4$, but we require $A \neq \pm 2$ in (6), so this cannot be the case.

11.9 Montgomery curve group law

The transformation of a Montgomery curve to Weierstrass form is a linear transformation that preserves the symmetry about the y-axis, so the geometric view of the group law remains the same: three points on a line sum to zero, which is is the point at infinity. To add points P_1 and P_2 we construct the line $\overline{P_1P_2}$ (using a tangent when $P_1 = P_2$), find the third intersection point with the curve, and then reflect over the y-axis to obtain $P_3 = P_1 + P_2$. In this section we compute explicit algebraic formulas for this operation, just as we did for curves in Weierstrass form earlier in the course.

The cases involving inverses and the point at infinity are easy (we have $P - P = 0$ and $P + 0 = 0 + P = P$), so let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be two (possibly equal but not opposite) affine points on the curve whose sum $P_3 = (x_3, y_3)$ we wish to compute. We first compute the slope m of the line $\overline{P_1P_2}$.

$$m = \begin{cases} \dfrac{y_1 - y_2}{x_1 - x_2} & \text{if } P_1 \neq P_2, \\ \dfrac{3x_1^2 + 2Ax_1 + 1}{2By_1} & \text{if } P_1 = P_2. \end{cases} \tag{7}$$

Now we want to intersect the line $y - y_1 = m(x - x_1)$ with the curve equation (6). Substituting $m(x - x_1) + y_1$ in for y, we get

$$B\left(m(x - x_1) + y_1\right)^2 = x^3 + Ax^2 + x. \tag{8}$$

We know x_1, x_2, and x_3 are the three roots of this cubic equation, since P_1, P_2, and $-P_3$ all lie on the curve and the line $\overline{P_1P_2}$. Thus the coefficient of x^2 in (8) must be equal to $x_1 + x_2 + x_3$. We get a Bm^2x^2 term on the left side of (8) and an Ax^2 term on the right, so we have $x_1 + x_2 + x_3 = Bm^2 - A$. Solving for x_3 and using the equation for $\overline{P_1P_2}$ to compute $-y_3$, we obtain

$$\begin{aligned} x_3 &= Bm^2 - (A + x_1 + x_2) \\ y_3 &= m(x_1 - x_3) - y_1. \end{aligned} \tag{9}$$

These formulas closely resemble the formulas for a curve in short Weierstrass form, but with an extra B and A in the equation for x_3. However, they have the key property that

they allow us to completely eliminate the y-coordinate from consideration. This is useful because the y-coordinate is not needed in many applications; we not need to know the y-coordinate of a point P in order to determine whether $mP = 0$ for a given integer m. This makes the y-coordinate superfluous in applications such as ECM and ECPP.

Let us consider the doubling case first. Plugging in the expression for m given by (7) in the case $P_1 = P_2 = (x_1, y_1)$ into (9) and remembering the curve equation $By^2 = x^3 + Ax^2 + x$,

$$\begin{aligned} x_3 &= B\frac{(3x_1^2 + 2Ax_1 + 1)^2}{4B^2y_1^2} - (A + 2x_1) \\ &= \frac{(3x_1^2 + 2Ax_1 + 1)^2 - 4(A + 2x_1)(x_1^3 + Ax_1^2 + x_1)}{4(x_1^3 + Ax_1^2 + x_1)} \\ &= \frac{(x_1^2 - 1)^2}{4x_1(x_1^2 + Ax_1 + 1)}, \end{aligned}$$

thus we can derive x_3 from x_1 without needing to know y_1. In projective coordinates,

$$\begin{aligned} &= \frac{(x_1^2 - z_1^2)^2}{4x_1z_1(x_1^2 + Ax_1z_1 + z_1^2)} \\ &= \frac{(x_1^2 - z_1^2)^2}{4x_1z_1\big((x_1 - z_1)^2 + (A + 2)x_1z_1\big)}. \end{aligned}$$

Thus we may write

$$\begin{aligned} x_3 &= (x_1 + z_1)^2(x_1 - z_1)^2 \\ 4x_1z_1 &= (x_1 + z_1)^2 - (x_1 - z_1)^2 \\ z_3 &= 4x_1z_1\big((x_1 - z_1)^2 + C(4x_1z_1)\big). \end{aligned} \tag{10}$$

where $C = (A + 2)/4$. Notice that these formulas do not involve y_1 and they only require 5 multiplications: 3 to compute x_3, none to compute $4x_1z_1$, and 2 more to compute z_3. One of these is a multiplication by the constant C, which may take negligible time if we can arrange for C to be small.

Now let us do the same thing for addition:

$$\begin{aligned} x_3 &= B\frac{(y_1 - y_2)^2}{(x_1 - x_2)^2} - (A + x_1 + x_2) \\ x_3(x_1 - x_2)^2 &= B(y_1 - y_2)^2 - (A + x_1 + x_2)(x_1 - x_2)^2 \\ &= By_1^2 + By_2^2 - 2By_1y_2 - (A + x_1 + x_2)(x_1 - x_2)^2 \\ &= -2By_1y_2 + 2x_1x_2(A + x_1 + x_2) + x_1 + x_2 \\ &= -2By_1Y_2 + x_2(x_1^2 + Ax_1 + 1) + x_1(x_2^2 + Ax_2 + 1) \\ &= -2By_1y_2 + \frac{x_2}{x_1}By_1^2 + \frac{x_1}{x_2}By_2^2 \\ &= B\frac{(x_2y_1 - x_1y_2)^2}{x_1x_2} \end{aligned} \tag{11}$$

This gives us an equation for x_3 in $P_3 = P_1 + P_2$, but it still involves the y-coordinates of P_1 and P_2. To address this, let us also compute the x-coordinate x_4 of $P_4 = P_1 - P_2$. The hard work is already done, we just need to negate y_2 in the equation for x_3. Thus

$$x_4(x_1 - x_2)^2 = B\frac{(x_2y_1 + x_1y_2)^2}{x_1x_2}. \tag{12}$$

Multiplying equations (11) and (12) yields

$$\begin{aligned} x_3x_4(x_1-x_2)^4 &= \frac{B^2(x_2^2y_1^2-x_1^2y_2^2)^2}{x_1^2x_2^2} = \frac{(x_2^2By_1^2-x_1^2By_2^2)^2}{x_1^2x_2^2)} \\ &= \frac{\left(x_2^2(x_1^3+Ax_1^2+x_1)-x_1^2(x_2^3+Ax_2^2+x_2)\right)^2}{x_1^2x_2^2} \\ &= \left(x_2(x_1^2+Ax_1+1)-x_1(x_2^2+Ax_2+1)\right)^2 \\ &= (x_2x_1^2-x_1x_2^2+x_2-x_1)^2 \\ &= \left((x_1-x_2)(x_1x_2-1)\right)^2. \end{aligned}$$

Canceling a factor of $(x_1-x_2)^2$ from both sides gives

$$x_3x_4(x_1-x_2)^2 = (x_1x_2-1)^2, \tag{13}$$

which does not involve y_1 or y_2 (but does require us to know x_4).

We now switch to projective coordinates:

$$\begin{aligned} \frac{x_3}{z_3}\cdot\frac{x_4}{z_4}\left(\frac{x_1}{z_1}-\frac{x_2}{z_2}\right)^2 &= \left(\frac{x_1x_2}{z_1z_2}-1\right)^2 \\ \frac{x_3}{z_3} &= \frac{z_4}{x_4}\cdot\frac{(x_1x_2-z_1z_2)^2}{(x_1z_2-x_2z_1)^2}, \end{aligned}$$

which yields

$$\begin{aligned} x_3 &= z_4\left[(x_1-z_1)(x_2+z_2)+(x_1+z_1)(x_2-z_2)\right]^2 \\ z_3 &= x_4\left[(x_1-z_1)(x_2+z_2)-(x_1+z_1)(x_2-z_2)\right]^2 \end{aligned} \tag{14}$$

These formulas require just 6 multiplications, but they assume that we already know the x-coordinate x_4/z_4 of P_1-P_2. But if we structure the double-and-add algorithm for scalar multiplication appropriately, we can use the formulas in (10) and (14) to efficiently compute the x-coordinate of the scalar multiple mP using what is known as a *Montgomery ladder*. We assume points are represented simply as projective pairs $(x:z)$ that omit the y-coordinate.

Algorithm 11.11 (Montgomery Ladder)**.**
Input: A point $P=(x_1:z_1)$ on a Montgomery curve and a positive integer m.
Output: The point $mP=(x_m:z_m)$.

1. Let $m=\sum_{i=0}^{k} m_i2^i$ be the binary representation of m.
2. Set $Q[0]=P$ and compute $Q[1]=2P$ (note that $P=Q[1]-Q[0]$).
3. For $i=k-1$ down to 0:
   a. $Q[1-m_i]\leftarrow Q[1]+Q[0]$ (Using $P=Q[1]-Q[0]$)
   b. $Q[m_i]\leftarrow 2Q[0]$
4. Return $Q[0]$.

The Montgomery ladder is the usual double-and-add algorithm, augmented to ensure that $Q[1] - Q[0] = P$ is invariant throughout. A nice feature of the algorithm is that every iteration of the loop is essentially the same: a Montgomery addition followed by a Montgomery doubling. This makes the algorithm resistant to side-channel attacks. If we assume that the input point P is in affine form $(x_1 : 1)$, then $z_1 = z_4 = 1$ in the addition formulas in (14), which saves one multiplication. This yields a total cost of $\big(10+o(1)\big)\log_2 m$ field multiplications for Algorithm 11.11, or only $\big(9+o(1)\big)\log_2 m$ if the constant C is small enough to make the multiplications by C negligible. This is faster than using Edwards' curves (at least in a side-channel resistant configuration where one is not using optimized doubling formulas).

An implementation of Algorithms 11.9 and 11.11 can be found in the Sage worksheet 18.783 Lecture 13 Montgomery ECM.sws.

11.10 Torsion on a Montgomery Curve

Every Montgomery point has $(0, 0)$ as a rational point of order 2 (as with curves in short Weierstrass form, the points of order 2 are precisely those with y-coordinate 0). This tells us that not every elliptic curve can be put in Montgomery form, since not every elliptic curve has a rational point of order 2. In fact, more is true.

Theorem 11.12. *The Montgomery curve E/k defined by $By^2 = x^3 + Ax^2 + x$ has either three rational points of order 2 or a rational point of order 4 (possibly both).*

Proof. The cubic $x^3 + Ax^2 + x$ has either one or three rational roots, and these roots are distinct, since the curve is nonsingular. If it has three roots, then there are three rational points of the form $(x, 0)$, all of which have order 2.

If it has only one root, then $x^2 + Ax + 1$ has no roots, so $A^2 - 4 = (A + 2)(A - 2)$ is not a quadratic residue. Therefore one of $A + 2$ and $A - 2$ is a quadratic residue (and the other is not), so either $\frac{A+2}{B}$ or $\frac{A-2}{B}$ is a quadratic residue. We will use this fact to find a point of order 4 that doubles to the 2-torsion point $(0, 0)$, which is the unique point on the curve whose x-coordinate is 0.

To get $x_3 = 0$ in the doubling formulas (10), we must have $x_1 = \pm z_1$, equivalently, $x_1/z_1 = \pm 1$. Plugging this into the curve equation, we seek a solution to either $By^2 = A+2$ or $By^2 = A - 2$. But we have already shown that either $\frac{A+2}{B}$ or $\frac{A-2}{B}$ is a quadratic residue, so one of these equations has a solution and there is a rational point of order 4. □

Thus, like Edwards curves, the torsion subgroup of a Montgomery curve always has order divisible by 4. For the purposes of the ECM algorithm this is actually a feature, since it slightly increases the likelihood that the group order will be smooth. In fact, most implementations use specific parameterizations to generate curves $E/\mathbb{Q}$ that are guaranteed to have even larger torsion subgroups, typically isomorphic to either $\mathbb{Z}/12\mathbb{Z}$ or $\mathbb{Z}/2\mathbb{Z}\oplus\mathbb{Z}/8\mathbb{Z}$; see [1, 4, 16] for examples (the $\mathbb{Z}/12\mathbb{Z}$ case is illustrated in the example implementation).

The converse of Theorem 11.12 does not hold; there are elliptic curves with three rational points of order 2 that cannot be put in Montgomery form. However, every elliptic curve with a rational point of order 4 can be put in Montgomery form.

Theorem 11.13. *Let $E\colon y^2 = x^3 + ax + b$ be an elliptic curve over a field k. Suppose $E(k)$ contains a point P of order 4, and let $2P = (x_0, 0)$. Then $3x_0^2 + a$ is a square in k and E can be put in Montgomery form $E'\colon By^2 = x^3 + Ax^2 + x$ by setting $B = 1/\sqrt{3x_0^2 + a}$ and $A = 3x_0B$; the map $(x, y) \mapsto (B(x - x_0), By)$ defines an isomorphism from E to E'.*

Proof. Let $P = (u, v)$. From the elliptic curve doubling formula, we have

$$\begin{aligned} x_0 &= \left(\frac{3u^2 + a}{2v}\right)^2 - 2u \\ &= \frac{(9u^4 + 6au^2 + a^2) - 8u(u^3 + au + b)}{4(u^3 + au + b)} \\ &= \frac{u^4 - 2au^2 - 8bu + a^2}{4(u^3 + au + b)}. \end{aligned}$$

Therefore u satisfies

$$u^4 - 4x_0u^3 - 2au^2 - (4ax_0 + 8b)u - 4bx_0 + a^2 = 0.$$

We have $0^2 = x_0^3 + ax_0 + b$, so we can replace b by $-x_0^3 - ax_0$, yielding

$$u^4 - 4x_0u^3 - 2au^2 + (8x_0^3 + 4ax_0)u + 4x_0^4 + 4ax_0^2 + a^2 = 0.$$

The LHS is a perfect square. If we put $u = z + x_0$ we can write this as

$$(z^2 - (3x_0^2 + a))^2 = 0.$$

Now $z = u - x_0 \in k$, so $z^2 - (3x_0^2 + a)$ must have a root in k. Thus $3x_0^2 + a$ is a square, as claimed, and it is nonzero because x_0 is not a repeated root of $x_0^3 + ax_0 + b$. Now let $B = 1/\sqrt{3x_0^2 + a}$ and $A = 3x_0B$ be as in the theorem and let $E' \colon By^2 = x^3 + Ax^2 + x$.

To check that $(x, y) \mapsto (B(x - x_0), By)$ defines an isomorphism from $E \to E'$, we plug $(B(x - x_0), By)$ into the equation for E' and note that

$$\begin{aligned} B(By)^2 &= (B(x - x_0))^3 + A(B(x - x_0))^2 + B(x - x_0) \\ B^2y^2 &= B^2(x^3 - 3x_0x^2 + 3x_0^2x - x_0^3) + 3x_0B^2(x^2 - 2x_0x + x_0^2) + x - x_0 \\ y^2 &= x^3 - 3x_0^2x + 2x_0^3 + (x - x_0)(3x_0^2 + a) \\ y^2 &= x^3 + ax - x_0^3 - ax_0 \\ y^2 &= x^3 + ax + b. \end{aligned}$$

This also shows that E' is not singular, since E is not (so we must have $A^2 \neq 4$). □

References

[1] A.O.L. Atkin and F. Morain, *Finding suitable curves for the elliptic curve method of factorization*, Mathematics of Computation **60** (1992), 399–405.

[2] R. Barbulescu, P. Gaudry, A. Joux, E. Thomé, *A heuristic quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic*, in Advances in Cryptology — EUROCRYPT 2014, LNCS **8441** (2014), Springer, 1–16.

[3] D.J. Bernstein, *How to find smooth parts of integers*, unpublished preprint, 2004.

[4] D.J. Bernstein, P. Birkner, T. Lange, and C. Peters, *ECM using Edwards curves*, Mathematics of Computation **82** (2013), 1139–1179.

[5] C. Bouvier, P. Gaudry, L. Imbert, H. Jeljeli, E. Thomé, *Discrete logarithms in $GF(p)$ — 180 digits*, NMBRTHRY listserv posting.

[6] E. Canfield, P. Erdős, and C. Pomerance, *On a problem of Oppenheim concerning "factorisatio numerorum"*, Journal of Number Theory **17** (1983), 1–28.

[7] A. Enge, *Discrete logarithms in curves over finite fields*, Finite fields and applications, Contemporary Mathematics **461**, AMS, 2008, 119–139.

[8] A. Enge and P. Gaudry, *A general framework for subexponential discrete logarithm algorithms*, Acta Arithmatica **102** (2002), 83–103.

[9] P. Gaudry, *Index calculus for abelian varieties of small dimension and the elliptic curve discrete logarithm problem*, J. Symbolic Computation **44** (2009), 1690–1702.

[10] R. Granger, T. Kleinjung, J. Zumbragel, *Discrete logarithms in $GF(2^{9234})$*, NMBRTHY listserv posting, 2014.

[11] D.M. Gordon, *Discrete Logarithms in $GF(p)$ using the number field sieve*, SIAM J. Discrete Math **6** (1993), 124–138.

[12] A. Granville, *Smooth numbers, computational number theory and beyond*, in Algorithmic Number Theory: Lattices, Number Fields, Curves and Cryptography (MSRI Workshop), Mathematical Sciences Research Institute Publications **44**, 2008, 267–324.

[13] A. Joux, *A new index calculus algorithm with complexity $L(1/4 + o(1))$ in very small characteristic*, in Selected Areas in Cryptography — SAC 2013, LNCS **8282** (2014), Springer, 355–379.

[14] A. Joux, R. Lercier, N. Smart, and F. Vercauteren, *The number field sieve in the medium prime case*, Advances in Cryptology — CRYPTO 2006, LNCS **4117** (2006), Springer, 326–344.

[15] H. Lenstra, *Factoring integers with elliptic curves*, Annals of Mathematics 126 (1987), 649–673

[16] P.L. Montgomery, *Speeding the Pollard and elliptic curve methods of factorization*, Mathematics of Computation **48** (1987), 243–264.

[17] J.M. Pollard, *Theorems of Factorization and Primality Testing*, Proceedings of the Cambridge Philosophical Society 76 (1974): 521–528

[18] L.C. Washington, *Elliptic Curves: Number Theory and Cryptography*, second edition, Chapman and Hall/CRC, 2008.

[19] P. Zimmermann and B. Dodson, *20 years of ECM*, Algorithmic Number Theory 7th International Symposium (ANTS VII), LNCS 4076 (2006), 525–542.

18.783 Elliptic Curves

Spring 2015