11.1 A generic lower bound for the discrete logarithm problem

We now give a lower bound for solving the discrete logarithm problem with a generic group algorithm. We will show that if p is the largest prime divisor of N, then any generic group algorithm for the discrete logarithm problem must use $\Omega(\sqrt{p})$ group operations. In the case that the group order $N = p$ is prime this bound is tight, since we have already seen that the problem can be solved with $O(\sqrt{N})$ group operations using the baby-steps giant-steps method. This lower bound applies not only to deterministic algorithms, but also to randomized algorithms. A generic Monte Carlo algorithm for the discrete logarithm problem must use $\Omega(\sqrt{p})$ group operations in order to be correct with probability greater than $1/2$, and the expected running time of any generic Las Vegas algorithm for the discrete logarithm problem is $\Omega(\sqrt{p})$ group operations.

The following theorem is due to Shoup [7]. It generalizes an earlier result of Nechaev [6] to a wider class of algorithms that includes all the methods we have seen for computing discrete logarithms. Our presentation here differs slightly from Shoup's and gives a sharper bound, but the essential details are the same. Recall that in our generic group model, each group element is uniquely represented as a bit-string via an injective identification map $\mathrm{id}\colon G \to \{0,1\}^n$, where $n = O(\log |G|)$.

Theorem 11.1 (Shoup)**.** *Let $G = \langle \alpha \rangle$ be group of order N. Let $\mathcal{B}$ be a black box for G supporting the operations* `identity`, `inverse`, *and* `compose`, *using a random identification map* $\mathrm{id}\colon G \to \{0,1\}^n$. *Let $\mathcal{A}\colon \{0,1\}^n \times \{0,1\}^n \to \mathbb{Z}/N\mathbb{Z}$ be a randomized generic group algorithm that makes at most $m - 4\lceil \log_2 N \rceil$ calls to $\mathcal{B}$, for some integer m, and let x denote a random element of $\mathbb{Z}/N\mathbb{Z}$. Then*

$$\Pr_{x,\mathrm{id},\tau}[\mathcal{A}(\mathrm{id}(\alpha), \mathrm{id}(x\alpha)) = x] < \frac{m^2}{2p},$$

where τ denotes the random coin-flips made by $\mathcal{A}$ and p is the largest prime factor of N.

Note that $\mathcal{A}$ can generate random elements of G by computing $z\alpha$ for random $z \in \mathbb{Z}/N\mathbb{Z}$ (we assume that $\mathcal{A}$ is given the group order N). The theorem includes deterministic algorithms as the special case where $\mathcal{A}$ does not use any bits of τ. Bounding the number of calls $\mathcal{A}$ makes to $\mathcal{B}$ might appear to preclude Las Vegas algorithms, but we will derive a corollary that addresses this.

Proof. To simply the proof, we will replace $\mathcal{A}$ by an algorithm $\mathcal{A}'$ that does the following:

1. Use $\mathcal{B}$ to compute $\mathrm{id}(N\alpha) = \mathrm{id}(0)$.
2. Simulate $\mathcal{A}$, using $\mathrm{id}(0)$ to replace `identity` operations, to get $y = \mathcal{A}(\mathrm{id}(\alpha), \mathrm{id}(x\alpha))$.
3. Use $\mathcal{B}$ to compute $\mathrm{id}(y\alpha)$.

In the description above we assume that the inputs to $\mathcal{A}$ are $\mathrm{id}(\alpha)$ and $\mathrm{id}(x\alpha)$; the behavior of $\mathcal{A}'$ when this is not the case is irrelevant. Note that steps 1 and 3 each require at most $2\lceil \log_2 N \rceil$ calls to $\mathcal{B}$ using double-and-add, so $\mathcal{A}'$ makes at most $m - 2$ calls to $\mathcal{B}$.

Let $\gamma_1 = \mathrm{id}(\alpha)$ and $\gamma_2 = \mathrm{id}(x\alpha)$. Without loss of generality we may assume that every interaction between $\mathcal{A}'$ and $\mathcal{B}$ is of the form $\gamma_k = \gamma_i \pm \gamma_j$, with $1 \le i, j < k$, where γ_i and γ_j

are group element identifiers that were either inputs or values previously returned by $\mathcal{B}$ (here the notation $\gamma_i \pm \gamma_j$ means that $\mathcal{A}'$ is using $\mathcal{B}$ to add or subtract the group elements identified by γ_i and γ_j). Note that $\mathcal{A}'$ can invert γ_j using $\gamma_i = \mathrm{id}(0)$.

Clearly the number of such interactions is a lower bound on the number of calls made by $\mathcal{A}'$ to $\mathcal{B}$. To further simplify matters, we will assume that the execution of $\mathcal{A}'$ is padded with operations of the form $\gamma_k = \gamma_1 + \gamma_1$ as required until k reaches m.

Let $N = p^e M$ with $p \perp M$. Define $F_k = a_k X + b_k \in \mathbb{Z}/p^e\mathbb{Z}[X]$ and $z_k \in \mathbb{Z}/M\mathbb{Z}$ via:

$$\begin{array}{lll} F_1 = 1 & z_1 = 1 & \\ F_2 = X & z_2 = x \bmod M & \\ \vdots & \vdots & \\ F_k = F_i \pm F_j & z_k = z_i \pm z_j & (\text{where } \gamma_k = \gamma_i \pm \gamma_j) \\ \vdots & \vdots & \\ F_m = F_i \pm F_j & z_m = z_i \pm z_j & (\text{where } \gamma_m = \gamma_i \pm \gamma_j) \end{array}$$

Note that $F_k(x) = \log_\alpha \gamma_k \bmod p^e$ for all k (think of X as the unknown value $x \bmod p^e$).

Now consider the following game, which models the execution of $\mathcal{A}'$. At the start of the game we set $F_1 = 1$, $F_2 = X$, $z_1 = 1$, and set z_2 to a random element of $\mathbb{Z}/M\mathbb{Z}$. We also set γ_1 and γ_2 to distinct random values in $\{0,1\}^n$. For rounds $k = 2, 3, \ldots, m$, the algorithm $\mathcal{A}'$ and the black box $\mathcal{B}$ play the game as follows:

1. $\mathcal{A}'$ chooses a pair of integers i and j, with $1 \le i, j < k$, and a sign $\pm$ that determines $F_k = F_i \pm F_j$ and $z_k = z_i \pm z_j$, and asks $\mathcal{B}$ for the value of γ_k.

2. $\mathcal{B}$ sets $\gamma_k = \gamma_\ell$ if $F_k = F_\ell$ and $z_k = z_\ell$ for some $\ell < k$, and otherwise $\mathcal{B}$ sets γ_k to a random element bit-string in $\{0,1\}^n$ that is distinct from γ_ℓ for all $\ell < k$.

After m rounds we pick $t \in \mathbb{Z}/p^e\mathbb{Z}$ at random and say that $\mathcal{A}'$ wins if $F_i(t) = F_j(t)$ for any $F_i \neq F_j$, otherwise $\mathcal{B}$ wins. The value x plays no role in the game, γ_2 is chosen at random.

We now claim that

$$\Pr_{x,\mathrm{id},\tau}[\mathcal{A}(\mathrm{id}(\alpha), \mathrm{id}(x\alpha)) = x] \le \Pr_{t,\mathrm{id},\tau}[\mathcal{A}' \text{ wins the game}], \tag{1}$$

where the id function on the right represents an injective map $G \to \{0,1\}^n$ that is compatible with the choices made by $\mathcal{B}$ during the game, i.e. there exists a sequence of group elements $\alpha = \alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_m$ such that $\mathrm{id}(\alpha_i) = \gamma_i$ and $\alpha_k = \alpha_i \pm \alpha_j$, where i, j, and the sign $\pm$ correspond to the values chosen by $\mathcal{A}'$ in the kth round.

For every value of x, id, and τ for which $\mathcal{A}(\mathrm{id}(\alpha), \mathrm{id}(x\alpha)) = x$, there is a value of t, namely $t = x \bmod p^e$, for which $\mathcal{A}'$ wins the game (here we use the fact that $\mathcal{A}'$ always computes $y\alpha$, where $y = \mathcal{A}(\mathrm{id}(\alpha), \mathrm{id}(x\alpha))$). The number of possible values of t is no greater than the number of possible values of x, hence (1) holds.

We now bound the probability that $\mathcal{A}'$ wins the game. Consider any particular execution of the game, and let $F_{i,j} = F_i - F_j$. We claim that for all i and j such that $F_{i,j} \neq 0$,

$$\Pr_t[F_{i,j}(t) = 0] \le \frac{1}{p}. \tag{2}$$

We have $F_{i,j}(X) = aX + b$ for some $a, b \in \mathbb{Z}/p^e\mathbb{Z}$ with a and b not both zero. Let p^d be the largest power of p that divides both a and b. Let $\overline{a} = a/p^d \bmod p$ and $\overline{b} = b/p^d \bmod p$.

Then $\overline{F} = \overline{a}X + \overline{b}$ is not the zero polynomial in $\mathbb{F}_p[X]$. Therefore $\overline{F}$ has at most one root and we have $\Pr[\overline{F}(t \bmod p) = 0] \leq 1/p$, which implies $\Pr[F_{i,j}(t) = 0] \leq 1/p$, proving (2).

If $\mathcal{A}'$ wins the game then there must exist an $F_{i,j} \neq 0$ for which $F_{i,j}(t) = 0$. Furthermore, since $F_{i,j}(t) = 0$ if and only if $F_{j,i}(t) = 0$, we may assume $i < j$. Thus

$$\begin{aligned}\Pr_{t,\mathrm{id},\tau}[\mathcal{A}' \text{ wins the game}] &\leq \Pr_{t,\mathrm{id},\tau}[F_{i,j}(t) = 0 \text{ for some } F_{i,j} \neq 0 \text{ with } i < j] \\ &\leq \sum_{i<j, F_{i,j}\neq 0} \Pr_t[F_{i,j}(t) = 0] \\ &\leq \binom{m}{2}\frac{1}{p} < \frac{m^2}{2p},\end{aligned}$$

where we have used a union bound ($\Pr[A \cup B] \leq \Pr(A) + \Pr(B)$) to obtain the sum. □

Corollary 11.2. *Let $G = \langle\alpha\rangle$ be a cyclic group of prime order N. Every deterministic generic algorithm for the discrete logarithm problem in G uses at least $(\sqrt{2} + o(1))\sqrt{N}$ group operations.*

The baby-steps giant-steps algorithm uses $(2 + o(1))\sqrt{N}$ group operations in the worst case, so this lower bound is tight up to a constant factor, but there is a slight gap. In fact the baby-steps giant-steps method is not optimal, the constant factor in the upper bound can be improved; see [1] (this still leaves a small gap).

Let us now extend Theorem 11.1 to our generic group model where the black box also supports the generation of random group elements for a cost of one group operation. We first note that having the algorithm generate random elements itself by computing $z\alpha$ for random $z \in \mathbb{Z}/N\mathbb{Z}$ does not change the lower bound significantly if only a small number of random elements are used, which applies to all of the algorithms we have considered.

Corollary 11.3. *Let $G = \langle\alpha\rangle$ be a cyclic group of prime order N. Every generic Monte Carlo algorithm for the discrete logarithm problem in G that uses $o(\sqrt{N}/\log N)$ random group elements uses at least $(1 + o(1))\sqrt{N}$ group operations.*

This follows immediately from Theorem 11.1, since a Monte Carlo algorithm is required to succeed with probability greater than 1/2. In the Pollard-ρ algorithm, assuming it behaves like a truly random walk, the number of steps required before the probability of a collision exceeds 1/2 is $\sqrt{2\log 2} \approx 1.1774$, so there is again only a slight gap in the constant factor between the lower bound and the upper bound.

In the case of a Las Vegas algorithm, we can obtain a lower bound by supposing that the algorithm terminates as soon as it finds a non-trivial collision (in the proof, this corresponds to a nonzero $F_{i,j}$ with $F_{i,j}(t) = 0$). Ignoring the $O(\log N)$ additive term, this occurs within m steps with probability at most $m^2/(2p)$. Summing over m from 1 to $\sqrt{2p}$ and supposing that the algorithm terminates in exactly m steps with probability $(m^2 - (m-1)^2)/(2p)$, the expected number of steps is $2\sqrt{2p}/3 + o(\sqrt{p})$.

Corollary 11.4. *Let $G = \langle\alpha\rangle$ be a cyclic group of prime order N. Every generic Las Vegas algorithm for the discrete logarithm problem in G that generates an expected $o(\sqrt{N}/\log N)$ random group elements uses at least $(2\sqrt{2}/3 + o(1))\sqrt{N}$ expected group operations.*

Here the constant factor $2\sqrt{2}/3 \approx 0.9428$ in the lower bound is once again only slightly smaller than the constant factor $\sqrt{\pi/2} \approx 1.2533$ in the upper bound given by the Pollard-ρ algorithm (under a random walk assumption).

Even if we have a generic algorithm that generates a large number of random elements, say $R = N^{1/3+\delta}$ for some $\delta > 0$ the cost of computing $z\alpha$ for R random values of z can be bounded by $2R + O(N^{1/3})$. If we let $n = \lceil \lg N/3 \rceil$ and precompute $c\alpha$, $c2^n\alpha$, and $c2^{2n}\alpha$ for $c \in [1, 2^n]$, we can then compute $z\alpha$ for any $z \in [1, N]$ using just 2 group operations. We thus obtain the following corollary, which applies to every generic group algorithm for the discrete logarithm problem.

Corollary 11.5. *Let $G = \langle \alpha \rangle$ be a cyclic group of prime order N. The expected number of group operations used by any generic algorithm for the discrete logarithm problem is $\Omega(\sqrt{N})$.*

In fact we can say that the constant factor is at least $\sqrt{2}/2$.

11.2 Index calculus

Having seen several examples of generic algorithms for the discrete logarithm problem, we now consider a *non-generic* algorithm for the discrete logarithm problem in the multiplicative group of a finite field, using a method known as *index calculus*. The same technique can be applied in other settings,[1] but we will restrict our attention to the simplest case: a finite field $\mathbb{F}_p$ of prime order. If α is a generator for $\mathbb{F}_p^*$ (a primitive root) then the discrete logarithm of $\beta \in \mathbb{F}_p^*$ with respect to α is also called the *index* of β (with respect to α), which explains the term "index calculus".

Given a positive integer B we define the *factor base* P_B as the set

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

The integer B is a *smoothness* bound; integers whose prime factors all lie in P_B are said to be *B-smooth*. For example, 1001 is 13-smooth.

Let us identify $\mathbb{F}_p \simeq \mathbb{Z}/p\mathbb{Z}$ with the set of integers in $[0, N]$, where $N = p - 1$ (note: this is precisely where we "cheat", since we are identifying group elements with integers in a particular way; this is not something that a generic algorithm is allowed to do). Now suppose that $\alpha^e\beta^{-1} \in [1, N]$ is B-smooth, say

$$\alpha^e\beta^{-1} = \prod_{i=1}^{b} p_i^{e_i},$$

where the e_i are nonnegative integers, most of which are zero. We can then write

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

as a linear equation in $b + 1$ variables $x_1, x_2, \ldots, x_{b+1}$ with the solution $x_i = \log_\alpha p_i$, for $1 \le i \le b$, and $x_{b+1} = \log_\alpha \beta$. If we collect $b + 1$ such equations, say, by choosing random values of e and discarding those for which $\alpha^e\beta^{-1}$ is not B-smooth, we can solve the resulting linear system. The system will often be under-determined; indeed, a particular variable x_i might not appear in any of the equations. 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.[2]

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

[1] These include groups associated to curves over finite fields; see [3] for a survey. But so far as we know this does not include the case of elliptic curves, other than in some rare special cases.

[2] 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 to succeed.

Algorithm 11.6 (Index calculus in a prime field $\mathbb{F}_p$)**.**

1. Pick a smoothness bound B and construct the factor base $P_B = \{p_1, \ldots, p_b\}$.
2. Generate $b+1$ random relations $R_i = (e_{i,1}, e_{i,2}, \ldots, e_{i,b}, 1, e_i)$ by picking $e \in [1, N]$ at random and and attempting to factor $\alpha^e \beta^{-1} \in [1, N-2]$ 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 $R_1, \ldots, R_{b+1}$ for x_{b+1} using linear algebra (e.g., row reduce the corresponding matrix).
4. If $x_{b+1} = \log_\alpha \beta$ is 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 11.7. 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 11.8. The relations $R_1, \ldots, R_{b+1}$ will be *sparse* (have few nonzero entries). The linear algebra step can be accelerated by using algorithms that take advantage of this fact.

Remark 11.9. While solving the system $R_1, \ldots, R_{b+1}$ one is likely to encounter zero divisors in the ring $\mathbb{Z}/N\mathbb{Z}$ ($N = p-1$ is even, so 2 is always a zero divisor). When this happens one can use a gcd computation to obtain a non-trivial factorization $N = N_1 N_2$ with N_1 and N_2 relatively prime. The system is then solved 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 if necessary).

Remark 11.10. 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 $p_i \in P_B$. Note that the x_i do not 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.

To choose B, we first note that α^e, and therefore $\alpha^e\beta^{-1}$, is uniformly distributed over $\mathbb{F}_p^*$, which we have identified with the set of integers in $[1, N]$. 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 cost of finding relations (let us suppose for the moment that the cost of the linear algebra step is negligible by comparison). In order to do this, we need to know the probability that a random integer in the interval $[1, N]$ is B-smooth.

11.3 Smooth numbers

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 [2] 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 and many other interesting facts about smooth numbers, we recommend the survey article by Granville [4].

11.4 Optimizing the smoothness bound

Let us assume that generating relations dominates the overall complexity of Algorithm 11.6, and that we simply use trial-division to factor $\alpha^e\beta^{-1}$ over P_b. Then the expected running time of Algorithm 11.6 is approximately

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

where b is the size of our factor base, $N = |\alpha| = p-1$, and $u = \log N/\log B$. The four factors correspond, respectively, to: (1) the number of relations we need, (2) the expected number of random exponents e we need to get one B-smooth integer $m = \alpha^e\beta^{-1} \in [1, N]$, (3) the number of trial divisions required to determine whether m is B-smooth (and factor it over P_B), and (4) the time for each trial division. By the Prime Number Theorem, we have $b = \pi(B) \sim B/\log B$. If we ignore logarithmic factors, we can replace $b+1$ and b by B and drop the $\mathsf{M}(\log N)$ factor.

Thus we wish to choose u to minimize the quantity

$$B^2 u^u = N^{2/u} u^u.$$

Taking logarithms, it suffices to minimize the function

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

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

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

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

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

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 $0 < a < 1$ the bound $L_N[a,c]$ 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

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

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 easy to parallelize.

Remark 11.11. As noted in Remark 11.10, if we are computing many (say at least $L_N[1/2,\sqrt{2}/]$) 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^2u^u. This yields an average expected running time of $L_N[1/2\sqrt{2}]$ per β.

A simple version of Algorithm 11.6 using trial-division for smoothness testing is implemented in the Sage worksheet

https://hensel.mit.edu:8002/home/pub/7/.

Finally, we note that using the elliptic curve factorization method (ECM) which we will discuss in the next lecture, the asymptotic cost of factoring smooth integers can be substantially reduced. One finds that with ECM the optimal smoothness bound becomes $B = L_N[1/2,1/\sqrt{2}]$, yielding an expected running time of $L_N[1/2,\sqrt{2}]$.[3]

But this is not the end of the story. Using more advanced techniques (analogous to those used in the number field sieve for factoring integers), one can achieve a heuristic running time of

$$L_N[1/3,\sqrt[3]{64/9}]$$

for computing discrete logarithms in $\mathbb{F}_p^*$. In fields of small characteristic it may be possible to do even better. There is a very recent result of Antoine Joux made public just within the past few weeks that claims to achieve $L_N[1/4+o(1),c]$ for computing discrete logarithms in fields $\mathbb{F}_{p^m}$, where p is very small (e.g. 2) and m is composite [5].

[3] As in Remark 11.11, when computing many discrete logarithms with respect to the same base using ECM, we should use $B = L_N[1/2,1/2]$, yielding an average expected running time of $L_N[1/2,1]$ per β.

References

[1] D. J. Bernstein and T. Lange, *Two grumpy giants and a baby*, Algorithmic Number Theory 10th International Symposium (ANTS X), 2012, to appear, preprint at http://math.ucsd.edu/~kedlaya/ants10/bernstein/paper.pdf.

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

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

[4] 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.

[5] A. Joux, *A new index calculus algorithm with complexity* $L(1/4 + o(1))$ *in very small characteristic*, preprint at http://eprint.iacr.org/2013/095.pdf.

[6] V. I. Nechaev, *Complexity of a determinate algorithm for the discrete logarithm*, Mathematical Notes **55**:2 (1994), 165–172.

[7] V. Shoup, *Lower bounds for discrete logarithms and related problems*, Proceedings of Eurocrypt '97 (1997), 256-266, revised version available at http://www.shoup.net/papers/dlbounds1.pdf.