ISSN (Print): 2476-8316
ISSN (Online): 2635-3490
Dutse Journal of Pure and Applied Sciences (DUJOPAS), Vol. 5 No. 1a June 2019

Improvements in Computing Discrete Logarithms in Finite Fields

[1]*Ali Maianguwa Shuaibu, [2]Sunday Babuba & [3]James Andrawus
[1, 2,3 Department] of Mathematics,
Federal University Dutse,
Jigawa State
Email: shuaibuali16@gmail.com

Abstract
The discrete logarithm problem forms the basis of numerous cryptographic systems. The most efficient attack on the discrete logarithm problem in the multiplicative group of a finite field is via the index calculus. In this paper, we examined a non-generic algorithm that uses index calculus. If a is a generator for $F_p^\times$ then the discrete logarithm of $\beta \in F_p^\times$ with respect to α is also called the index of β (with respect to α), whence the term index calculus. This algorithm depends critically on the distribution of smooth numbers (integers with small prime factors), which naturally leads to a discussion of two algorithms for factoring integers that also depend on smooth numbers: the Pollard $p-1$method and the elliptic curve method. In conclusion, we suggest improved elliptic curve as well as hyperelliptic curve cryptography methods as fertile ground for further research.

Keywords: Elliptic curve, Index calculus, Multiplicative group, Smooth numbers

Introduction

It is known that the logarithm $log_b a$ is a number x such that $b^x = a$, for given numbers a and b. Similarly, in any group G, powers b^k can be defined for all integers k, and the discrete logarithm$log_b a$ is an integer k such that $b^k = a$. In number theory,$x = ind_r a(mod\, m)$ is read as the index of a to the base r modulo m which is equivalent to $r^x = a(mod\, m)$ if r is the primitive root of m and greatest common divisor, $gcd(a, m) = 1$.

Discrete logarithms are easily computable in some special cases. However, no efficient method is known for computing them in general. Several important algorithms in public-key cryptography base their security on the assumption that the discrete logarithm problem over carefully chosen groups has no efficient solution.

One of the simplest settings for discrete logarithms is the group$(Z_p)^\times$. This is the group of multiplication modulo the prime p. Its elements are congruence classes modulo p, and the group product of two elements may be obtained by ordinary integer multiplication of the elements followed by reduction modulo p.

*Author for Correspondence

The kth power of one of the numbers in this group may be computed by finding its kth power as an integer and then finding the remainder after division by p. When the numbers involved are large, it is more efficient to reduce modulo p multiple times during the computation. Regardless of the specific algorithm used, this operation is called modular exponentiation.

For example, consider $(\mathbf{Z}_{17})^{\times}$. To compute 3^4 in this group, compute $3^4 = 81$, and then divide 81 by 17, obtaining a remainder of 13. Thus $3^4 = 13$ in the group $(\mathbf{Z}_{17})^{\times}$.

The discrete logarithm is just the inverse operation. For example, consider the equation $3^k \equiv 13 \pmod{17}$ for k. From the example above, one solution is $k = 4$, but it is not the only solution. Since by Fermat's little theorem $3^{16} \equiv 1 \pmod{17}$, it also follows that if n is an integer then $3^{4+16n} \equiv 3^4 \times (3^{16})^n \equiv 13 \times 1^n \equiv 13 \pmod{17}$. Hence the equation has infinitely many solutions of the form $4 + 16n$. Moreover, because 16 is the smallest positive integer m satisfying $3^m \equiv 1 \pmod{17}$, these are the only solutions. Equivalently, the set of all possible solutions can be expressed by the constraint that $k \equiv 4 \pmod{16}$.

The discrete logarithm problem is considered to be computationally intractable. That is, no efficient classical algorithm is known for computing discrete logarithms in general.

A general algorithm for computing $log_b\, a$ in finite groups G is to raise b to larger and larger powers k until the desired a is found. This algorithm is referred to as trial multiplication. It requires running time linear in the size of the group G and thus exponential in the number of digits in the size of the group. Therefore, it is an exponential-time algorithm, practical only for small groups G.

More sophisticated algorithms exist, usually inspired by similar algorithms for integer factorization. These algorithms run faster than the naïve algorithm, some of them linear in the square root of the size of the group, and thus exponential in half the number of digits in the size of the group. However, none of them run in polynomial time in consideration of the number of digits in the size of the group.

In the following sections, Index calculus, Pollard $p-1$ and elliptic curve factorization methods for computing discrete logarithms are critically examined.

Index calculus

Index calculus is a method for computing discrete logarithms in the multiplicative group of a finite field. It might not seem directly relevant to the elliptic curve discrete logarithm problem, but when we discuss pairing-based cryptography, these two problems are easily unrelated (Sutherland, 2017). Moreover, the same approach can be applied to elliptic curves over nonprime finite fields, as well as abelian varieties of higher dimension (Enge, 2008; Gaudry, 2009). Here, our attention is restricted to the simplest case which involve a finite field of prime order p. Let us consider$F_P \simeq \mathbb{Z}/p\,\mathbb{Z}$ with our usual choice of representatives for $\mathbb{Z}/p\,\mathbb{Z}$, integers in the interval $[0, N]$, with $N = p - 1$. This simple statement is precisely what will allow us to do something that a generic algorithm cannot. It enables us lift elements of $F_p \simeq \mathbb{Z}/p\,\mathbb{Z}$ to the ring $\mathbb{Z}$ and consider their prime factorizations, something that makes no sense in a multiplicative group in which every element is a unit, 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 to obtain relations between the discrete logarithms of various elements of $F_p^{\times}$.

Let us fix a generator α for $F_p^{\times}$, and let $\beta \in \langle\alpha\rangle$ 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 it gives

$$\prod p_i^{e_i} = \alpha^e\beta^{-1} \qquad (2)$$

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

$$\sum e_i \log_\alpha p_i + \log_\alpha \beta = e \tag{3}$$

which determines $\log_\alpha \beta$ as a linear expression in the discrete logarithms$\log_\alpha p_i$. Note that the$p_i$are viewed both as primes in $\mathbb{Z}$ and elements of $F_p^\times \simeq (\mathbb{Z}/p\,\mathbb{Z})^\times$). This does not immediately give us the desired result, since we do not know the values of $\log_\alpha p_i$. However, if we repeat this procedure using many different values of e, we may obtain a system of linear equations that we can then solve for $\log_\alpha \beta$.

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

$$P_B = \{p: p \le B \text{ is prime }\}\{p_1, p_2, \ldots, p_b\}, \tag{4}$$

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 $\alpha^e B^{-1} \in [1, N] \subseteq \mathbb{Z}$ that we can write as a product of primes in our factor base P_B, in fact most would not. But some choices will work, and for those that do we obtain a linear equation of the form

$$e_1 \log_\alpha p_1 + e_2 \log_\alpha p_2 + \ldots + e_b \log_\alpha p_b = e, \tag{5}$$

Many of the e_i may be zero. We do not yet know any of the discrete logarithms on the left hand side, but we can view

$$e_1 x_1 + e_2 x_2 + \ldots + e_b x_b + x_{b+1} = e \tag{6}$$

as a linear equation in $b+1$variables $x_1, x_2, \ldots, x_{b+1}$ 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, the resulting linear system may determine a unique value x_{b+1}, which represent the discrete logarithm we wish to compute. This system will typically be under-determined; indeed, some variables x_i may not appear in any of our equations. But it is quite likely that the value of x_{b+1}, which is present in every equation, will be uniquely determined. We will not attempt to prove this, because 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 (Sutherland, 2017).

Sutherland (2017) suggests the following algorithm to compute $\log_\alpha \beta$.

Algorithm 1 (Index calculus in a prime field F_p).

1. Pick a smoothness bound B and construct the factor base $P_B = \{p_1, p_2, \ldots, p_b\}$.
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 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 $R_1, \ldots, R_{b+1}$ 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 let us first note the following.

Remark 1. 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 2. The relations, $R_1, \ldots, R_{b+1}$ will typically be very sparse (have few nonzero entries), which allows one to accelerate the linear algebra step significantly.

Remark 3. 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 Chinese Remainder Theorem (CRT) to recover the value of x_{b+1} in $\mathbb{Z}/N\mathbb{Z}$ (recurse as necessary).

Remark 4. Solving the system of relations will determine not only $x_{b+1} = log_\alpha \beta$, but also many $x_i = log_\alpha p_i$ for $p_i \in P_B$, which 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 are 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.

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. However, it also makes it more difficult to determine whether this is indeed 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 at 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). If we choose $e \in [1, N]$ uniformly at random then α^e, and therefore $\alpha^e\beta^{-1}$, will be uniformly distributed over $F_p^\times$, which have been identified with the set of integers in $[1, N]$. In order to obtain an optimal value of B, there is need to know the probability that a random integer in the interval $[1, N]$ is B-smooth (Sutherland, 2017).

Optimizing the smoothness bound

Assume that generating relations dominates the overall complexity of Algorithm 1, and we use trial-division to attempt to factor $\alpha^e\beta^{-1}$ over P_B. Then the expected running time of Algorithm 1 is approximately

$$(b + 1).u^u.b.M(log\, N) \tag{7}$$

where $u = log\, N / log\, B$. The four factors in (6) are:

i. $b + 1$: the number of relations R_i that we need;
ii. u^u: the expected number of random exponents e needed to obtain a B-smooth integer $m = \alpha^e\beta^{-1} \in [1, N]$;
iii. b: the number of trial divisions to test whether m is B-smooth and factor it if it is;
iv. $M\, log\, N$: the time for each trial division.

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

$$B^2u^u = N^{2/u}u^u, \tag{8}$$

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

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

and 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

$$u = 2\sqrt{\log N / \log\log N}, \tag{9}$$

then

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

as desired. The choice of u in (8) implies that we should use the smoothness bound

$$B = N^{\frac{1}{u}} = exp(\frac{1}{u}\log N)$$

$$= exp(\frac{1}{2}\sqrt{\log N \log\log N}$$

We have used the following standard sub-exponential asymptotic notation

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

Also note that

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

is polynomial in $\log N$, whereas

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

is exponential in $\log N$. For $0 < \alpha < 1$ the bound $L_N[\alpha, c]$ is said to be sub-exponential. We also have

$u^u = exp(u \log u) = L_N[\frac{1}{2}, 1]$, thus the total expected running time is

$$B^2u^u = L_N[\tfrac{1}{2}, \tfrac{1}{2}]^2.L_N[\tfrac{1}{2}, 1] = L_N[\tfrac{1}{2}, 2].$$

The cost of the linear algebra step is certainly no worse than $\tilde{O}(b^3)$.We may bound this by $\tilde{O}(B^3)$, which in our sub exponential notation is $L_N[\frac{1}{2}, \frac{3}{2}]$. 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 $\tilde{O}(b^2)$. However, in large computations the linear algebra step can become a limiting factor because it is memory intensive and not as easy to parallelize as relation finding (Sutherland, 2017).

As noted earlier, if we are computing many (say at least $L_N[\frac{1}{2}, \frac{\sqrt{2}}{2}]$) 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 $B = N^{\frac{1}{u}}$ to minimize Bu^u rather than B^2u^u. This yields an average expected running time of $L_N[\frac{1}{2}, \sqrt{2}]$ per β.

Improvements

Using the elliptic curve factorization method (ECM) described in the next section, the cost of testing and factoring B-smooth integers can be made sub-exponential in B and polynomial in

$log\ N$. This effectively changes B^2u^u in (8) to Bu^u, and the optimal smoothness bound becomes $B = L_N[\frac{1}{2}, \frac{1}{\sqrt{2}}]$, yielding a heuristic expected running time of $L_N[\frac{1}{2}, \sqrt{2}]$. There is a batch smoothness testing algorithm due to Bernstein (2004)for a sufficiently large set of integers yields an average time per integer that is actually polynomial in $log\ N$, but this does not 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 the form

$$L_N[\frac{1}{3}, c]$$

for computing discrete logarithms in $F_p^{\times}$ (again using an index calculus approach); see (Gordon, 1993).

Infinite fields of small characteristic $F_{p^n} \simeq F_P[x]/(f(x))$, one uses the function field sieve, where the factor base now consists of low degree polynomials in $F_P[x]$ that represent elements of F_p^n when reduced modulo $f(x)$. This also yields an $L_N[\frac{1}{3}, c]$ bound (with a smaller value of c), and it is now known that such a bound holds (under heuristic assumptions) for all finite fields (Joux *et al*, 2006).

But this is far from the end of the story. In 2013 Antoine Joux announced an index calculus approach for finite fields of the form F_{q^k} with $q \approx k$ that heuristically achieves an $L_N[\frac{1}{4} + o(1), c]$ time complexity (Joux, 2014). Shortly thereafter a recursive variant of Joux's approach was used to obtain a heuristically quasi-polynomial-time complexity of $k^{O(log\ k)}$, which in terms of $N = q^k$ is bounded by $L_N[\varepsilon, c]$ for every $\varepsilon, c > 0$. At first glance the assumption q ≈ k might seem restrictive, but even for finite fields of the form F_{2^k} with k prime it suffices to compute discrete logarithms in the extension field $F_{2^{kr}}$ with $r = [lgk]$, which for $q = 2^r \approx k$ has the desired form F_{q^k}, and even though we are now working in a larger field, the $k^{O(log\ k)}$ bound is still quasi-polynomial in the input size k, and as a function of $N = 2^k$ it is dominated by $L_N[\varepsilon, c]$ for all $\varepsilon, c > 0$. The current record for computing discrete logarithms in finite fields of characteristic 2 was set in the field $F_{2^{9234}}$ in 2014 (not long after Joux's result was announced), using about 45 core-years of computation time (Granger *et al*, 2014). The record for prime degree finite fields was set the same year in the field$F_{2^{1279}}$, using less than 4 core years (Kleinjung, 2014). By contrast the largest discrete logarithm computation over a safe prime fieldF_p (one in which $(p-1)/2$ is prime), was set in 2016 using a 768-bit prime p and took approximately 6600 core years (Kleinjung *et al*, 2016). The recent dramatic improvements in computing discrete logarithms in finite fields of small characteristic has effectively eliminated any interest in pairing-based elliptic curve cryptography over such fields. As discussed in Sutherland (2017), in pairing-based cryptography one needs to consider the difficulty of the discrete logarithm problem both in the group of rational points on an elliptic curve over a finite field F_q and in the multiplicative group of a low degree extension of F_q. None of these results have had any impact on prime fields.

The Pollard $p-1$ method

Now we want to consider algorithms for factoring integers that also rely on smooth numbers. In 1974, Pollard (1974) introduced a Monte Carlo algorithm for factoring integers 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 2 (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$\ell \leq B$:
   a. Set $b = b^{\ell^e} \, mod \, N$, where $\ell^{e-1} < N \leq \ell^e$.
   b. If$d = gcd(b-1, N)$ is not 1 then return$d$ if$d < N$ or *failure* if $d = N$.
4. Return

It should be noted that instead of using a fixed bound B, we could simply allow the algorithm to 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 another algorithm if the $p-1$ method fails. In any case, it is convenient to have B fixed for the purposes of analysis.

Example

Let $N = 899$and suppose we pick $a = 2$in step 1. Then $d = 1$ in step 2, and the table below illustrates the situation at the end of each iteration of step 3.

Table 1: Pollard $p-1$ factorization

ℓ	e	b	d
2	10	605	1
3	7	690	1
5	5	683	31

The algorithm finds the factor 31 of $N = 29.31$ when$\ell = 5$ because $\#(\mathbb{Z}/31)^\times = 30 = 2.3.5$ is 5-smoothbut $\#(\mathbb{Z}/29)^\times = 28 = 2^2.7$ is not: if we put $m = 2^{10}.3^7.5^5$ then m is divisible by $\#(\mathbb{Z}/31\mathbb{Z})^\times$ but not by$\#(\mathbb{Z}/29\mathbb{Z})^\times$, and it follows that we always have$a^m \equiv 1 \, mod \, 3\,1$, but for most choices of a we will have $a^m \not\equiv 1 \bmod 29$, leading to $d = gcd(a^m - 1, 29.31) = 31$.

If we had instead used $N = 31.41$ we would have found $d = N$ when $\ell = 5$ and failed because $\#(\mathbb{Z}/41\mathbb{Z})^\times = 40 = 2^3.5$ has the same largest prime factor as $\#(\mathbb{Z}/31\mathbb{Z})^\times$.

Theorem 1.

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 $\ell_p \leq B$ and $\ell_p < \ell_q$ then Algorithm 2 succeeds with probability at least $1 - \frac{1}{\ell_q}$.

Proof.

If $a \equiv 0 \, mod \, p$ 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 $m = \prod_{\ell \leq \ell_p} \ell^\ell$ is a multiple of $p-1$. By Fermat's little theorem, $b = a^m \equiv 1 \, mod \, p$ 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 almost all values of N, Algorithm 2 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 Sutherland (2017), 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 Algorithm 2 can factor N within a few seconds; no other algorithm is known that has any hope of factoring N in any reasonable amount of time. However, in the worst-case the running time is $O(\pi(B)\, M(logN)\, log\, N)$, and with $B = \sqrt{N}$ the complexity is $O(\sqrt{N}\, M(logN))$, the same as 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 moduli), 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? For a given value of B, the expected time for the algorithm to achieve a success is

$$\frac{O(\pi(B)\, M(logN)\, log\, N)}{Pr[success]} \tag{10}$$

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 exactly the 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.

By the Canfield-Pomerance-Erdős Theorem (Sutherland, 2017), 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), this easily simplify (10) to

$$N^{1/u} u^u. \tag{11}$$

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

$$B = N^{1/u} = exp\big((1/\sqrt{2} + o(1))\sqrt{logNloglog\, N}\big) = L_N[\tfrac{1}{2}, \tfrac{1}{\sqrt{2}}],$$

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$u^u = exp(\, u\, log\, u) = L_N[\tfrac{1}{2}, \tfrac{1}{\sqrt{2}}]$, and the total expected running time is therefore

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

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) sub-exponential running time. But this does not help much if there is a particular integer$N$ that we want to factor.

The elliptic curve method for factoring integers

By the use of elliptic curves one can effectively achieve the randomized scenario envisioned above where N is fixed. The Pollard $p-1$ works in the group $(\mathbb{Z}/N\mathbb{Z})^\times$, which, by the CRT, we can think of as a simultaneous computations in the groups $(\mathbb{Z}/p\mathbb{Z})^\times$ for primes $p \mid N$; 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 modulo N, we have an opportunity to factor N if $E(F_p)$ has smooth order, for some prime $p \mid N$. Now, the main difference is that the curve E can be varied while keeping N constant. This results in a new groups $E(F_p)$ each time we change E. This is the basis of the elliptic curve method (ECM), introduced by Lenstra(1987).

The algorithm is essentially the same as Pollard's $p-1$ method. Rather than exponentiating a random element of $(\mathbb{Z}/N\mathbb{Z})^\times$ 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 does not happen we can simply switch to a different curve and try again.

As in Pollard's algorithm, we do not 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$mP = (\mathbb{Q}_x : \mathbb{Q}_y : \mathbb{Q}_z)$ is a multiple of P that reduces to 0 modulo a prime p dividing N, then p divides $gcd(\mathbb{Q}_z, N) \neq N$. 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 as to keep the coordinates reduced modulo N throughout the algorithm.

In order to get a proper divisor of N, we also need $gcd(\mathbb{Q}_z, N) \neq N$. 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 $gcd(\mathbb{Q}_z, N) = N$ 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. 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. 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 (this is useful for smoothness testing).

Algorithm 3. **(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 $\mathbb{Q} = P = (x_0 : y_0 : 1)$.

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 $\mathbb{Z}/N\mathbb{Z}$.

Conclusion

The complexity of computing discrete logarithms in prime fields is of substantial practical interest, since it provides an estimate of the size of the prime that has to be used.

In this paper, we critically examined three algorithms for factoring integers. Algorithm 3 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's curves are an excellent choice; see (Bernstein *et al*, 2013) for a detailed discussion of how to efficiently implement ECM using Edwards curves. Another popular choice is Montgomery curves (Montgomery, 1987); as explained in Bernstein and Lange (2017), there is a close relationship between Montgomery curves and Edwards's curves. 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.

There is still a wealth of research areas in elliptic curve cryptography to delve into. Particularly, we suggest improved elliptic curve as well as hyper-elliptic curve cryptography methods as fertile ground for further research.

References

Bernstein, D.J. (2004). How to find smooth parts of integers, unpublished preprint.

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

Bernstein, D.J. and Lange, T. (2017).Montgomery curves and the Montgomery ladder, Cyrptology ePrint Archive, Report 2017/293.

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

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

Granger, R., Kleinjung, T. and Zumbragel, J.(2014). Discrete logarithms in$GF(2^{9234})$, NMBRTHY listserv posting, January 31.

Enge, A. (2008). Discrete logarithms in curves over finite fields, Finite fields and applications, Contemporary Mathematics 461, AMS, 119–139.

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

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

Kleinjung, T., Diem, C., Lenstra, A.K., Priplata, C., and Stahlke, C.(2016). Discrete logarithms in GF(p) — 768 bits, NMBRTHRY listserv posting, June 16.

Kleinjung, T.(2014).Discrete logarithms in $GF(2^{1279})$,NMBRTHRYlistervposting,October 17.

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

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

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

Sutherland, A.(2017).Math.mit.edu/classes/18.783/2017/LectureNotes11.pdf, March 15.