Several Algorithms Using Elliptic Curves

Chris Phillips

December 2018

1 Introduction

This paper will outline the steps and provide justification for two algorithms related to elliptic curves, namely Lenstra's Factorization Algorithm and Schoof's Algorithm for counting the number of points on an elliptic curve over a finite field $\mathbb{F}_p$.

2 Lenstra's Factorization Algorithm

Lenstra's Algorithm is essentially an improvement on Pollard's $p-1$ Algorithm. We will begin with a discussion of Pollard's Algorithm to motivate the use of elliptic curves.

2.1 Pollard's $p-1$ Algorithm

First remember Fermat's Little Theorem: If p is a prime integer and a is any integer not divisible by p we have

$$a^{p-1} \equiv 1(\text{mod } p).$$

Let n be the composite integer we want to factor and suppose n has a prime factor p such that $p-1$ is a product of small primes. Fermat's Little Theorem implies that p divides $gcd(a^{p-1}-1,n)$. This is the crucial observation for the algorithm.

Without knowing p computing $a^{p-1}-1$ is impossible so make a reasonable guess. Choose $k = 2^{e_2} \cdot 3^{e_3} \cdot ... \cdot r^{e_r}$ where $\{2, 3, ..., r\}$ is the list of the first primes up to an arbitrarily chosen prime r and $\{e_2, e_3, ..., e_r\}$ are relatively small integers.

Using the fast powering algorithm compute $gcd(a^k-1,n)$. If $p-1|k$ then $p|a^k-1$, thus $g = gcd(a^k-1,n) \geq p > 1$. If $g \neq n$ then we have succeeded in finding a non-trivial factor of n. Otherwise a new a is chosen and we loop back through.

2.1.1 Pollard's Algorithm: Outline of Steps

Given $n \geq 2$ a composite integer to be factored.

1. Set $a = 2$.
2. Set $k = 2$ k_{max} = pre-specified upper bound. While $k \leq k_{max}$ loop:
   (a) Set $a = a^k \pmod{n}$.
   (b) Compute $g = gcd(a-1, n)$.
   (c) If $n > g > 1$, return g.
   (d) If $g = n$, Increment a, go to 2.
3. Increment k, go to 2.

   This is a reformatting of the outline found in [Jos15].

2.2 Lenstra's Algorithm

Notice that in order for Pollard's Algorithm to work without taking an extremely large amount of computations we need a prime factor of n such that $p-1$ is a product of small primes. We get this value $p-1$ because it is the order of $(\mathbb{Z}/p\mathbb{Z})^*$, the multiplicative group of integers mod p. This is a fairly large supposition when working with a random composite number. Lenstra's Algorithm replaces the group $(\mathbb{Z}/p\mathbb{Z})^*$ with the group of points on an elliptic curve over a finite field to side-step this problem.

Choose an elliptic curve E and a point $P \in E(\mathbb{F}_p)$. If the number of points on E over $\mathbb{F}_p$, $\#E(\mathbb{F}_p)$, happens to divide an integer k we know that $kP = \mathcal{O}$ where $\mathcal{O}$ is the identity in the group $E(\mathbb{F}_p)$. This will lead to a means of finding p.

At first glance the advantage of working in this setting might not be apparent, but working over $\mathbb{Z}/p\mathbb{Z}$ we are stuck with the order $p-1$ for our group. Working with the group of points on a curve over $\mathbb{F}_p$ we can change the curve. The number of points changes for a fixed p as we vary the curve so the odds of the algorithm succeeding increase.

2.2.1 Description of Lenstra's Algorithm

To factor a given composite integer n pick a pair (x_1, y_1) and b with x_1, y_1, b all integers modulo n then find c such that

$$c \equiv y_1^2 - x_1^3 - bx_1$$

then set $E : y^2 = x^3 + bx + c$. In this way we have chosen a curve, E, and point, P, to start with. By choosing the pair (x_1, y_1) before fixing a curve we have guaranteed that E has a point to work with, and we do not need to use any resources to compute said point.

Now we must compute kP for a given k. To do this we use a binary expansion trick similar to the fast powering algorithm. Start by writing

$$k = k_0 + 2k_1 + 2^2k_2 + 2^3k_3 + ... + 2^nk_n,$$

with $k_i = 0$ or 1. Then calculate

$$\begin{aligned}
P_0 &= P \\
P_1 &= 2P_0 = 2P \\
P_2 &= 2P_1 = 4P \\
&\vdots \\
P_n &= 2P_{n-1} = 2^n P
\end{aligned}$$

Then $kP =$ the sum of all P_i where $k_i \neq 0$. This will take at most $2\log_2 k$ steps of doubling and reduction modulo n[Jos15].

Because n is not prime we run into some difficulty with the usual addition and doubling formulas. Recall, to add points $Q_1 = (x_1, y_1), Q_2 = (x_2, y_2)$ such that $Q_1 + Q_2 = Q_3 = (x_3, y_3)$ we use the formulas

$$x_3 = \lambda^2 - x_1 - x_2, \; y_3 = -\lambda x_3 - (y_1 - \lambda x_1)$$

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1}.$$

Thus we can only add points if $(x_2 - x_1)$ is invertible modulo n. Which is not guaranteed because $\mathbb{Z}/n\mathbb{Z}$ is not a field. We run into a similar problem with the usual formulas for doubling where

$$\lambda = \frac{2x^2 + 2ax + b}{2y}.$$

In either case we consider what happens when trying to invert the denominator.

Let $a = 2y$ or $a = (x_2 - x_1)$ be the element in $\mathbb{Z}/n\mathbb{Z}$ we are trying to invert. There are three possible outcomes:

1. $gcd(a, n) = 1$, then a is invertible modulo n. We can proceed adding or doubling.

2. $gcd(a, n) = g$ where $1 < g < n$, we cannot add or double the points as a is not invertible modulo n, but we have succeeded in our original goal of finding a non-trivial factor of n. Rejoice!

3. $gcd(a, n) = n$ we cannot proceed with point addition or doubling, and we have not found a non-trivial factor. Pick a new curve and repeat the process.

2.2.2 Lenstra's Algorithm: Outline of Steps

Given a composite integer n to factor. First check that $gcd(6, n) = 1$ and that n is not a perfect power to rule out factors that would be easier to find.

1. Choose random $b, x_1, y_1 \in \mathbb{Z}/n\mathbb{Z}$.
2. Set $P = (x_1, y_1)$, $c \equiv y_1^2 - x_1^3 - bx_1 \pmod{n}$.
3. Set $E : y^2 = x^3 + bx + c$.
4. Set $d = 2$, d_{max} = pre-determined upper bound. While $d \leq d_{max}$ loop:
   (a) Compute $Q = dP$.
   (b) If (a) fails we have found $g > 1$ such that $g|n$.
   (c) If $g < n$, return g.
   (d) Else, $g = n$, go to 1. Choose new curve and point.
5. Increment d, go to 4.

This is a reformatting of the outline found in [Jos15]

2.2.3 Why Should This Algorithm Work?

If p, q are two primes that divide n, the number we want to factor, and we let $E : y^2 = x^3 + ax + b$ be an elliptic curve then we have that $y^2 = x^3 + ax + b \bmod n$ implies the same equation holds mod p and mod q. Because p and q are prime we know that the points on the curve over the finite fields $\mathbb{F}_p$ and $\mathbb{F}_q$ form groups with our usual methods for addition. These groups have order N_p and N_q respectively where N_p is $\#E(\mathbb{F}_p)$ and N_q is $\#E(\mathbb{F}_q)$. Thus if $kP = \mathcal{O}_p$ then $k|N_p$ where $P \in E(\mathbb{F}_p)$. Similarly, if $kP = \mathcal{O}_q$ then $k|N_q$ where $P \in E(\mathbb{F}_q)$. Here $\mathcal{O}_i$ is the point at infinity over $\mathbb{F}_i$.

Hasse's Theorem: For $E(\mathbb{F}_q)$, an elliptic curve over a finite field $\mathbb{F}_q$, the number of points on $E(\mathbb{F}_q)$ $\#E(\mathbb{F}_q) = q + 1 - \epsilon$ where $|\epsilon| \leq 2\sqrt{q}$.

Hasse's Theorem tells us that $\#E(F_p)$ is relatively close to $p + 1$ and similarly $\#E(\mathbb{F}_q)$ is close to $q + 1$. Thus, it is likely that we can find a k such that $kP = \mathcal{O}_p$ that $kP \neq \mathcal{O}_q$. When we find k kP will not be a point on the original curve with points mod n.

3 Schoof's Algorithm

Schoof's Algorithm is a polynomial time algorithm for counting the number of points on an elliptic curve over a finite field, $\mathbb{F}_q$. The algorithm finds the error term from Hasse's Theorem by solving for it modulo a set of small primes then using the Chinese Remainder Theorem to find the desired result. Schoof's Algorithm by itself is not efficient enough to be used for large q, but Elkies and Atkins have improved the algorithm to a version known as S.E.A.(Schoof, Elkies, Atkins).

3.1 The Algorithm

Hasse's Theorem: For $E(\mathbb{F}_q)$, an elliptic curve over a finite field $\mathbb{F}_q$, the number of points on $E(\mathbb{F}_q) = \#E(\mathbb{F}_q) = q + 1 - \epsilon$ where $|\epsilon| \leq 2\sqrt{q}$.

We can restate Hasse's Theorem to say

$$|q + 1 - \#E(\mathbb{F}_q)| \leq 2\sqrt{q}.$$

Let $t = q+1-\#E(\mathbb{F}_q)$ with the above inequality it is sufficient to find t modulo N where $N > 4\sqrt{q}$. Then $\#E(\mathbb{F}_q) = q + 1 - t$.

Solving for t mod N directly will be inefficient if N is relatively large. The idea comes from building a set $S = \{l_1, ..., l_n\}$ of small primes such that $\Pi_{i=1}^{n} l_i = N$ and solving for t mod l_i for all $l_i \in S$. Then we can use the Chinese Remainder Theorem to solve for t mod N.

3.1.1 Division Polynomials

Let $E : y^2 = x^3 + ax + b$ be a Weierstrass curve.

Here we introduce the division polynomial, $\Psi_n(x, y) \in \mathbb{F}_q[x, y]$ for $n \in \mathbb{Z}, n \geq 1$.

$$\begin{aligned}
\Psi_1(x, y) &= 1 \\
\Psi_2(x, y) &= 2y \\
\Psi_3(x, y) &= 3x^4 + 6ax^2 + 12bx - a^2 \\
\Psi_4(x, y) &= 4y(x^6 + 5ax^4 + 20bx^3 - 5a^2x^2 - 4abx - 8b^2 - a^3) \vdots \\
\Psi_{2n}(x, y) &= \Psi_n(\Psi_{n+2}\Psi_{n-1}^2) - \Psi_{n-2}\Psi_{n+1}^2)/2y \\
\Psi_{2n+1}(x, y) &= \Psi_{n+1}\Psi_n^3 - \Psi_{n+1}^3\Psi_{n-1}
\end{aligned}$$

For $n \geq 0$ $\Psi_n(x, y) = 0$ precisely when (x, y) is in the group of n-torsion points on E.

From these we can get an explicit formula for nP where $n \in \mathbb{Z}$, $P = (x, y) \in E(\mathbb{F}_q)$, such that $nP \neq \mathcal{O}$

$$nP = (x - \frac{\Psi_{n-1}\Psi_{n+1}}{\Psi_n^2}, \frac{\Psi_{n+2}\Psi_{n-1}^2 - \Psi_{n-2}\Psi_{n+1}^2}{4y\Psi_n^3}) \tag{1}$$

For later purposes we will eliminate y^2 terms using $y^2 = x^3 + ax + b$ and call the new polynomial Ψ'_n. Then Ψ'_n is either in $\mathbb{F}_q[x]$ or $y\mathbb{F}[x]$. Define

$$f_n(x) = \begin{cases} \Psi'_n(x, y)\text{if } n \text{ is odd} \\ \Psi'_n(x, y)/y \text{ if } n \text{ is even} \end{cases}$$

Note that the degree of f_n is

$$deg(f_n) = \begin{cases} \frac{1}{2}(n^2 - 1) \text{ if } n \text{ is odd} \\ \frac{1}{2}(n^2 - 4) \text{ if } n \text{ is even} \end{cases}$$

[Sch85]

3.1.2 The Frobenius Endomorphism

Let $t = q + 1 - \#E(\mathbb{F}_q)$ and

$$\phi : E(\mathbb{F}_q) \to E(\mathbb{F}_q),\ (x, y) \mapsto (x^q, y^q)$$

the q^{th} power Frobenius map.

The important takeaway from the Frobenius map is that it satisfies

$$\phi^2 - t\phi + q = 0 \in End(E)$$

where $End(E)$ is the ring of endomorphisms of E[Sil09].

3.1.3 Solving for t mod l

In order to guarantee the existence of l-torsion points we must work over a finite field extension of $\mathbb{F}_q$. Call this extension $\mathbb{F}_{\overline{q}}$.

Let l be an odd prime not equal to 2, and $P = (x, y) \in E(\mathbb{F}_{\overline{q}})[l]$ an- l-torsion point on E. Then

$$\phi^2(P) - t\phi(P) + qP = \mathcal{O}$$

$$(x^{q^2}, y^{q^2}) - t(x^q, y^q) + q(x, y) = \mathcal{O}$$

Because P is an l-torsion point we can reduce t, q mod l such that $t' \equiv t \bmod l$ and $q' \equiv q \bmod l$. Then we have

$$(x^{q^2}, y^{q^2}) - t'(x^q, y^q) + q'(x, y) = \mathcal{O}.$$

Now we have reduced the problem to finding a point in $E(\mathbb{F}_{\overline{q}})[l]$ which satisfies

$$t'(x^q, y^q) = (x^{q^2}, y^{q^2}) + q'(x, y).$$

These l-torsion points are defined in potentially large extension fields of $\mathbb{F}_q$ so, to avoid growing complexity, in this process we will work with the division polynomial $f_l(x)$. Recall, that the coordinates of l-torsion points are roots of this polynomial, and that it has degree $\frac{1}{2}(l^2 - 1)$. Perform all computations from now on in the quotient ring

$$R_l = \frac{\mathbb{F}_{\overline{q}}[x, y]}{f_l(x), y^2 - x^3 - ax - b}$$

Any time there is a non-linear term for y replace it with $y^2 = x^3 + ax + b$ and any time there is a power x^d with $d \geq \frac{1}{2}(l^2 - 1)$ divide by $f_l(x)$ and take the remainder. This reduces the maximal degree we are working with to $\frac{1}{2}(l^2 - 1) - 1 = \frac{1}{2}(l^2 - 3)$.

In this way computing t' for enough primes l we can find t via the Chinese Remainder Theorem.

3.1.4 Outline of Steps

Given $E : y^2 = x^3 + ax + b$, $\mathbb{F}_q$ a finite field for which the number of points is desired.

1. Set $A = 1$, $l = 3$, $q' = q \bmod l$.
2. While $A \leq 4\sqrt{q}$.
   (a) For $n = 0, 1, ..., l - 1$
      i. Working in R_l, if $(x^{q^2}, y^{q^2}) + q'(x, y) = n(x^q, y^q)$ break n-loop.
   (b) Set $t_l = n$, $A = A \cdot l$, $l =$ next largest prime.
3. Use Chinese Remainder theorem to find t using $t_3, t_5, ..., t_r$ where r is that prime necessary such that $3 \cdot 5 \cdot ... \cdot r \geq 4\sqrt{q}$.
4. Return $\#E(\mathbb{F}_q) = q + 1 - t$.

   This is a reformatting of the outline found in [Sil09]

3.1.5 Proof of Complexity

Theorem: Let $E(\mathbb{F}_q)$ be an elliptic curve over a finite field $\mathbb{F}_q$. Schoof's Algorithm is a polynomial time algorithm to compute $\#E(\mathbb{F}_q)$; In fact, it computes $\#E(\mathbb{F}_q)$ in $O((\log q)^8)$ steps.

We first prove three lemmas.

Lemma 1: The largest prime l used is $O(\log q)$

Proof. The prime number theorem states: If $\pi(x)$ is the number of primes less than or equal to x, then

$$\lim_{x\to\infty} \pi(x)\frac{\log x}{x} = 1$$

This is equivalent to the statement

$$\lim_{x\to\infty} \frac{1}{x} \sum_{l \leq x, l \text{ prime}} \log l = 1$$

Then

$$\prod_{l<x} l \approx e^x$$

Thus, in order to make the product $> 4\sqrt{q}$ it is sufficient to take x such that

$$\begin{aligned} e^x &> 4\sqrt{q} \\ (e^x)^2 = e^{2x} &> 16q \\ 2x &> \log(16q) \\ x &> \frac{1}{2}\log(16q) \end{aligned}$$

Thus it is sufficient to choose x such that $x \approx \frac{1}{2}\log(16q)$. □

Lemma 2: Multiplication in the ring R_l can be done in $O(l^4(\log q)^2)$ bit operations.

Proof. Elements in R_l are of degree $O(l^2)$. Multiplication of two such elements followed by reducing mod $f_l(x)$ takes $O(l^4)$ operations of addition and multiplication in $\mathbb{F}_q$. Multiplication in $\mathbb{F}_q$ takes $O((\log q)^2)$ bit operations. Thus, operations in R_l take $O(l^4(\log q)^2)$ bit operations. $\square$

Lemma 3: It takes $O(\log q)$ operations in R_l to reduce $x^q, y^q, x^{q^2}, y^{q^2}$ in the ring R_l.

Proof. Using the square and multiply algorithm described **2.2.1** we can compute (x^n, y^n) with $O(\log n)$ operations. Note that these values are only computed once and used throughout the algorithm. $\square$

Proof. (**Theorem**) From **Lemma 1** we know that the primes l used are less than $O(\log q)$ and that there are $O(\frac{\log q}{\log \log q})$ such ls used. Then the algorithm only loops through step 2., from the outline, that many times. The inner loop, (a), is run through at most l times, in other words less than $O(\log q)$ times. Lemma 2 tells us that multiplication in R_l takes $O(l^4(\log q)^2) = O((\log q)^6)$ operations. Solving for $t'(x^q, y^q)$ in (a) takes $O(1)$ operations using the previous value.

With all of this we get a total number of operations:

$$O(\log q) \cdot O(\log q) \cdot O((\log q)6) = O((\log q)^8).$$

$\square$

References

[Sch85] Rene Schoof. "Elliptic Curves Over Finite Fields and the Computing of Square Roots mod p". In: *Mathematics of Computation* 44.170 (1985), pp. 483–494.

[Sil09] Joseph H. Silverman. *The Arithmetic of Elliptic Curves.* Springer, 2009. ISBN: 9780387094939.

[Jos15] John T. Tate Joseph H. Silverman. *Rational Points on Elliptic Curves.* Undergraduate Texts in Mathematics. Springer, 2015. ISBN: 9783319185873.