WXML Final Report: Elliptic Curve Primality Test

Rohan Hiatt, Daria Mićović,
Blanca Viña Patiño, Bryan Tun Pey Quah
Supervisors: Amos Turchet, Travis Scholl

Spring 2017

1 Introduction

Prime numbers are fascinating objects in mathematics, fundamental to number theory and cryptography. Primality tests are algorithms that take in an integer as input and return whether that number is prime or composite. Most practical applications require primality tests to run relatively fast despite the input size; today, the largest known prime number has over twenty million digits so proving that a number of this size is prime can be computationally expensive.

The first deterministic primality test that also runs in polynomial time relative to the binary representation of the input was published in 2002. Although this algorithm, the AKS primality test, represents an important breakthrough in the field of computational number theory, it is seldom used in practice. Our objective this quarter was to determine why this idealized algorithm is not practical enough compared to other primality tests; more specifically, the Elliptic Curve Primality Test. Unlike AKS, this primality test is probabilistic which means that the algorithm does not guarantee that the output is entirely correct. We took our AKS implementation from last quarter as well as implemented ECPP and compared both algorithms in runtime and accuracy. Ultimaltely, we confirmed that probabilistic methods are still preferred due to faster execution times despite the small chance that the output is incorrect.

1.1 The initial problem

Our main motivation to study the Elliptic Curve Primality Proving (ECPP) algorithm was to compare its asymptotic behavior to other primality tests such as the AKS Primality test which we researched and studied the previous quarter. Although AKS is deterministic and runs in polynomial time, we wanted to see why it is not used in practice as well as determine why ECPP is a much better option. When testing crypto-sized primes, mathematicians and cryptographers prefer probabilistic tests when testing large numbers because they have faster runtimes than deterministic algorithms. For each primality testing algorithm, we studied and analyzed each individual step, as well as compared their execution times and overall accuracy.

2 Progress

For the first half of the quarter we focused on learning basic algebra and understanding arithmetic behind elliptic curves. Learning the elementary mathematics behind elliptic curves was important because we needed to understand point addition and counting for the elliptic curve primality test. We successfully implemented the double-and-add algorithm for torsion points as well as a general naive point addition algorithm that calculates the slope between two points and finds a third intersection. We also spent time developing our own version of Schoof's algorithm which determines the number of points on a given elliptic curve, but encountered many difficulties and have not yet been able to get it running properly. Finally, we implemented the Goldwasser-Kilian Elliptic Cuurve Primality Test and compared its performance to our implementations of AKS, trial division, and Fermat's Little Theorem.

2.1 Computational

2.1.1 Adding Points on Elliptic Curves

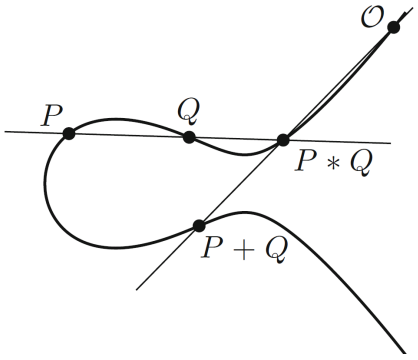

Figure 1: Point addition on Elliptic Curves [3]

Elliptic curves are of the form $y^2 = x^3 + Ax + B$ where A and B are constants such that $4A^3 + 27B^2 \neq 0$, with a special point $\mathcal{O}$ at infinity such that one can give a structure of a group to its points in which $\mathcal{O}$ is the identity element in the group. Geometrically, points P and Q on an elliptic curve are added by drawing a line between them and reflecting the point of the third intersection over the x-axis. From the geometric representation, we can derive the algebraic formulas which we implement in our code.

We begin with the arithmetic of adding two distinct points in an elliptic curve over a finite field, $E(\mathbb{F}_p)$. We have three cases: (1) where $-P, Q$ are distinct (and neither are equal to $\mathcal{O}$), (2) where one of P, Q is equal to $\mathcal{O}$, and (3) where $P = Q$, that is, we double P.

We first discuss the case where $-P, Q$ are distinct. The Silverman-Tate text provides the following formulas that derive a third point, $R := P \bigoplus Q$, where $\bigoplus$ is the notation for addition of points on an elliptic curve, given initial starting points (x_1, y_1) and (x_2, y_2):

$$x_3 = m^2 - x_1 - x_2 \text{ and } y_3 = -(mx_3 + b)$$

where m is the slope of line between the two points and b is the y-intersect of this point on the Cartesian plane. These are defined as:

$$m = \frac{y_2 - y_1}{x_2 - x_1} \text{ and } b = y_1 - mx_1 = y_2 - mx_2$$

For the second case, if we are adding a point (x_0, y_0) to itself, in other words doubling it, the slope m, where $y^2 = f(x)$, becomes:

$$m = \frac{f'(x_0)}{2y_0}$$

which is the implicit derivative of the elliptic curve for the point. Lastly, if adding a point P to the infinity point $\mathcal{O}$, we get $P \bigoplus \mathcal{O} = P$ [3].

2.1.2 Schoof's Algorithm

This algorithm determines the number of points on an elliptic curve over finite fields which we needed for the Elliptic Curve Primality Test. Schoof's main idea behind this algorithm is based on the Hasse bound:

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

which estimates that the number of points on an elliptic curve over $\mathbb{F}_q$ up to a bound where q is a prime integer.

The algorithm also utilizes the Frobenius endomorphism which maps a point to it's q-th power:

$$\pi : (x, y) \mapsto (x^q, y^q)$$

The Frobenius map has the characteristic equation

$$\pi^2 - t\pi + q = 0$$

where $t = q + 1 - |E(\mathbb{F}_q)|$ from here we can solve for the number of points, $|E(\mathbb{F}_q)|$. This is done by computing several t values modulo a set of prime numbers and then recovering the value of t using the Chinese Remainder Theorem.

2.1.3 Goldwasser-Kilian Elliptic Curve Primality Test

The Goldwasser-Kilian Elliptic Curve Primality Test uses randomly generated elliptic curves over $\mathbb{Z}/n\mathbb{Z}$ to reduce the size of the input to be less than 2^{16} so it can be tested by trial division. This bound is used because trial division is faster than the elliptic curve test for numbers less than 2^{16}. The

algorithm requires that the number of points on the elliptic curve be of the form kq where k is a small number and q is probably prime. In our implementation we used Fermat's Primality Test to check for primality at this step. Once we have a probably prime q, we can recursively plug it back into the algorithm until it gets tested by trial division. At the final step, the result of trial division indicates the primality of the initial input [4].

The Goldwasser-Kilian test is probabilitsic because it is not guaranteed that we will find an elliptic curve with the condition on the number of points. However, in practice this can be done quickly.

The Algorithm

Input: integer $n > 1$

1. If ($n \leq 2^{16}$ for $n \in \mathbb{N}$):
   Output result from trial division.
2. Randomly choose an elliptic curve over $\mathbb{Z}/n\mathbb{Z}$.
3. Compute $m = \#(E(\mathbb{Z}/n\mathbb{Z}))$ using Schoof's Algorithm.
4. If $m \neq kq$ where q is probably prime:
   GOTO Step 2
5. Pick a point P on the curve such that $mP = \mathcal{O}$ and $kP \neq \mathcal{O}$.
6. RESTART Step 1 with $n = q$.

2.2 Results

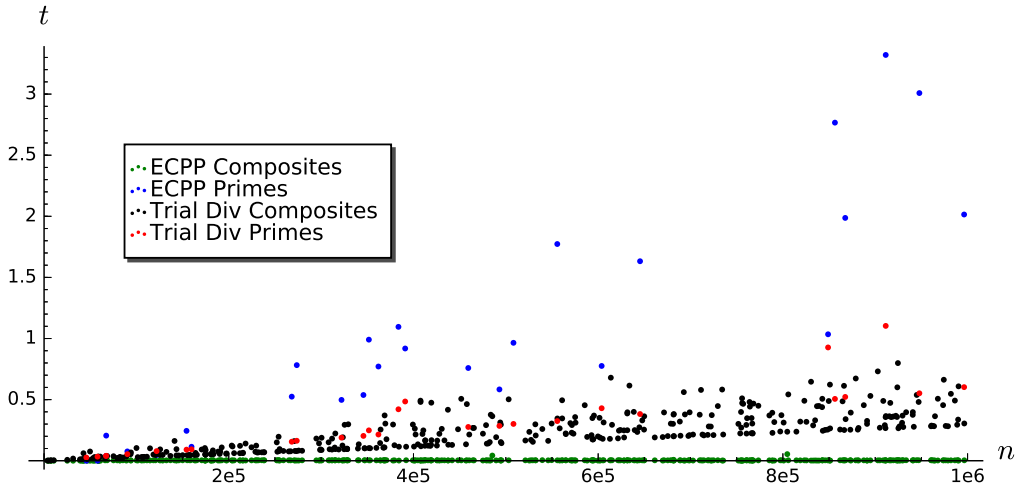

Figure 2: Graph of Number vs Time Taken in seconds

Figure 2 denotes a comparison between our implementation of both the ECPP and trial division algorithms. For numbers less than $2^{16} \approx 6e5$, ECPP utilizes trial division. From our graph, we can see that ECPP takes (slightly) more time than trial division does for these numbers - possibly due to added machine overhead for running ECPP or machine error.

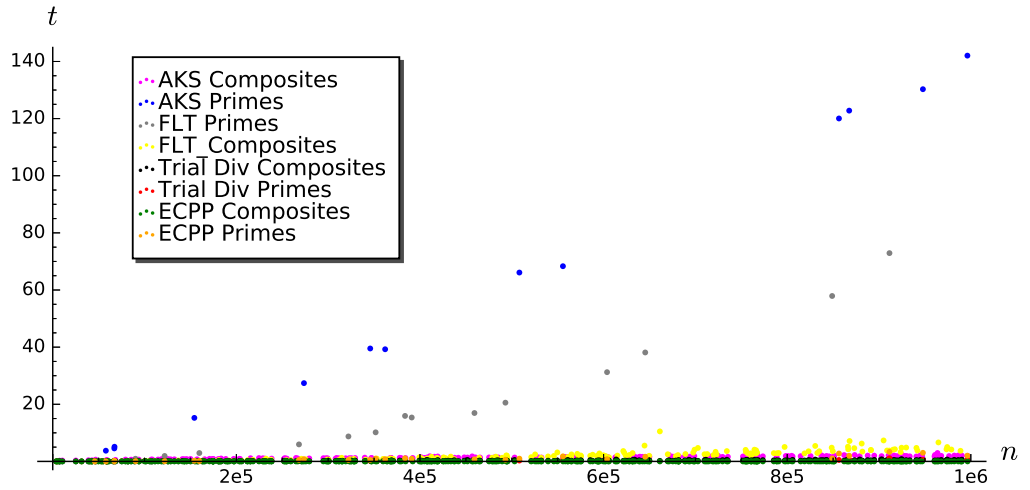

Figure 3: Graph of Number vs Time Taken in seconds

Figure 2 is a comparison of all 4 algorithms we have implemented for both prime and composite numbers. We can see that AKS takes the longest time among the algorithms to certify prime numbers and our implementation of a Fermat's Little Theorem Test takes the second longest time.

3 Future directions

One of our biggest challenges this quarter was implementing Schoof's algorithm on Cocalc. Schoof's algorithm was only ever designed to work with prime q's in $\mathbb{F}_q$. However, the ECPP uses the algorithm both when q is and is not prime, resulting in several issues. The first issue was that we needed to create a rational function class so that certain operations were supported in $\mathbb{F}_p$. This took a while to make and led us to change much of the code we already had. Another big issue we had was creating a recursive function for nth division polynomials so we started using the Cocalc implementation. However the built in function did not return the division polynomials we wanted. Ultimately, we were unable to get Schoof's algorithm working so we used the Cocalc built in function to return the number of points on an elliptic curve. In the future we want to spend time debugging Schoof's algorithm so that we can finally solve all issues associated with it. Additionally we would like to gather more data for larger numbers. The whole point of primality testing

is to be able to determine primality for crypto-sized integers. However, the tools we have now do not allow us to test such big numbers. Hopefully in the future we can get access to the resources neccessary to obtain bigger and better data.

References

[1] G. Musiker. Schoof's algorithm for counting points on elliptic curves over a finite field. 2005.

[2] SageMath, Inc. *SageMath, the Sage Mathematics Software System (Version 6.10)*, 2017. `http://www.sagemath.org`.

[3] J. H. Silverman and J. T. Tate. *Rational points on elliptic curves*. Undergraduate Texts in Mathematics. Springer, Cham, second edition, 2015.

[4] O. Uzunkol. Atkins ecpp (elliptic curve primality proving) algorithm. *Tech. University of Kaiserslautern*, 2004.