An Overview of Elliptic Curve Primality Proving

Frank Li

December 15, 2011

1 Introduction

Primes are of fundamental importance in number theory, and primality testing is one of the oldest problems in mathematics. Various algorithms have been presented over the past two millenia, ever since Eratosthenes detailed his eponymous sieve in 274 B.C. After important theoretical advances by Fermat, Euler, Legendre, and Gauss in the seventeeth and eighteenth centuries, the rise of computational approaches to primality testing began in the 1970's.

Early modern algorithms were closely linked to factoring, causing the algorithms to be slow or only work for numbers of special forms. Miller (1976) [11] gave a deterministic general-purpose polynomial-time algorithm, but his result assumed the unproven Extended Riemann Hypothesis. Solovay-Strassen (1977) [16] and Rabin (1980) [15] independently presented probabilistic polynomial-time primality tests. Adelman-Pomerance-Rumely (1983) [1] and Cohen-Lenstra (1984) [7] gave nearly-polynomial-time deterministic tests that did not rely on unproven assumptions. These tests could not prove that a number was prime; instead, they would generate either a proof of compositeness or conclude that the input was a probable prime.

In contrast, primality proving algorithms generate a certificate of primality, in which the primality of a large number is reduced to the primality of a smaller number. This forms a chain of primes that can then be verified in polynomial time. Pratt (1975) [14] first proved such certificates exist but gave no efficient implementation. Then Goldwasser-Kilian (1986) [8] exhibited a randomized algorithm that generates a primality certificate in expected polynomial time on almost all inputs. This approach used elliptic curves and laid the framework for elliptic curve primality proving (ECPP). Atkin (1986) [4] improved this result and Adleman-Huang (1992) [2] modified it to run in expected polynomial time on all inputs. More recently, Agrawal-Kayal-Saxena (2002) [3] resolved a long-standing open question by describing a deterministic polynomial-time proving algorithm, at last establishing that PRIMES is in P.

Of these algorithms, ECPP has seen the greatest success in proving the primality of random large numbers. Specialized tests such as the Lucas-Lehmer test and Fermat test have yielded the largest primes known, but these are all of a special form. For general numbers, the best AKS-class tests have yielded $\tilde{O}(\log^4 n)$ time algorithms [6], on par with the asymptotic heuristic bound on fast ECPP [12]. However, the constants in AKS-class tests are much higher than in ECPP, and in practice ECPP is the fastest known algorithm for proving the primality of general numbers.

This paper explores the inaugural ECPP algorithm presented by Goldwasser-Kilian [8] as well as later improvements on the algorithm.

2 Background

Goldwasser-Kilian's ECPP algorithm relies on a result from Lenstra [10], whose work on factoring using elliptic curves yielded insight into primality proving.

Theorem 1. (Distribution of orders of random elliptic curves [10]):
Let $p > 5$ be a prime, and let $S \subseteq \{p+1-\lfloor\sqrt{p}\rfloor, \ldots, p+1+\lfloor\sqrt{p}\rfloor\}$. Let A, B be randomly chosen from $\mathbb{F}_p$. Then there exists a fixed constant k such that,

$$\Pr(\#E_{A,B}(\mathbb{F}_p) \in S) > \frac{k}{\log p} \cdot \frac{|S|-2}{2\lfloor\sqrt{p}\rfloor+1}$$

In other words, if S is some subset of numbers in a small interval satisfying a certain property and $|S| > 2$, the order of a random elliptic curve is at least $O(1/\log p)$ times as likely to satisfy that property as a random element in this interval. Note that this interval is about half the Hasse interval. We will use this theorem in our analysis of the running time of the ECPP algorithm.

We now establish notation and recall a basic result for elliptic curve operations over the ring $\mathbb{Z}_n$.

Lemma 2. (Elliptic curve addition over $\mathbb{Z}_n$ [8]):
Let n be an integer not divisible by 2 or 3. Let $p > 3$ be a prime divisor of n, and let $4A^3 + 27B^2 \neq 0 \mod p$. For any $x \in \mathbb{Z}_n$, define $x_p = x \mod p \in \mathbb{F}_p$, and for any $L = (x, y) \in E_{A,B}(\mathbb{Z}_n)$, define $L_p = (x_p, y_p) \in E_{A,B}(\mathbb{F}_p)$ and $\infty_p = \infty \in E_{A,B}(\mathbb{F}_p)$. For any $L, M \in E_{A,B}(\mathbb{Z}_n)$, if $L + M$ is defined, then $(L+M)_p = L_p + M_p$.

The heart of Goldwasser-Kilian's approach is the following theorem:

Theorem 3. (Elliptic curve primality proof [9]):
Let n be an integer not divisible by 2 or 3. Let $A, B \in \mathbb{Z}_n$ such that $(4A^3 + 27B^2, n) = 1$, and let $L \neq \infty \in E_{A,B}(\mathbb{Z}_n)$. If there exists a prime $q > (\sqrt[4]{n}+1)^2$ such that $qL = \infty$, then n is prime.

Proof. By contradiction. Suppose n is composite. Then there exists a prime $p > 3$ dividing n, so that $p \leq \sqrt{n}$. Note that $4A^3 + 27B^2 \neq 0 \mod p$, since otherwise $(4A^3 + 27B^2, n) \neq 1$. Therefore $L_p \in E_{A,B}(\mathbb{F}_p)$, and $qL_p = (qL)_p = \infty_p = \infty$ by Lemma 2. So the order of L_p must divide q, and since $L_p \neq \infty$ and q is prime, it follows that that the order of L_p must equal q. But by Hasse's Theorem, the order of $L_p \leq \#E_{A,B}(\mathbb{F}_p) \leq (\sqrt{p}+1)^2 \leq (\sqrt[4]{n}+1)^2 < q$, a contradiction. So n must be prime. ■

3 Algorithm

The ECPP algorithm follows closely from Theorem 3. At a high level, the following algorithms are employed:

Algorithm 4. Generate-Curve(p):

1. Randomly select $A, B \overset{R}{\leftarrow} \mathbb{F}_p$ until $(4A^3 + 27B^2, p) = 1$, $p + 1 - \lfloor\sqrt{p}\rfloor \leq \#E_{A,B}(\mathbb{F}_p) \leq p + 1 + \lfloor\sqrt{p}\rfloor$, and $\#E_{A,B}(\mathbb{F}_p)$ is even. (Note: we may compute $\#E_{A,B}(\mathbb{F}_p)$ efficiently using Schoof's algorithm.)
2. Set $q = \#E_{A,B}(\mathbb{F}_p)/2$. If q is divisible by 2 or 3, retry from step 1.
3. Run a probabilistic primality test on q for $2\log p$ trials. If a trial ever returns ”composite,” retry from step 1. (Note: we may use the Miller-Rabin [15] or Solovay-Strassen [16] tests, or any other probabilistic test with an error rate that decreases exponentially as a function of $\log p$.)
4. Return $(A, B), q$.

Algorithm 5. Find-Point($p, q, (A, B)$):

1. Randomly select $x \overset{R}{\leftarrow} \mathbb{F}_p$ until $x^3 + Ax + B$ is a quadratic residue in $\mathbb{F}_p$.
2. Compute the square roots of $x^3 + Ax + B$ and set y to be one of the square roots randomly. Let $L = (x, y) \in E_{A,B}(\mathbb{F}_p)$.
3. If $qL \neq \infty$, retry from step 1.
4. Return L.

Algorithm 6. Prove-Prime(p):

1. Let $i := 0$, $p_0 = p$, and $\textsc{lowerbound} = \max(2^{(\log p)^{C/\log\log\log p}}, 37)$, where C is some constant such that the deterministic algorithm in step 3 will run in polynomial time in $\log p$.
2. While $p_i > \textsc{lowerbound}$, do:
   (a) $(A_i, B_i), p_{i+1} \leftarrow$ Generate-Curve(p_i).
   (b) $L_i \leftarrow$ Find-Point($p_i, p_{i+1}, (A, B)$)
   (c) $i := i + 1$.
   (d) If any of the p_i is divisible by 2 or 3, or if $i \geq (\log p)^{\log\log p}$, break out of the loop and retry from step 1.
3. Use a deterministic algorithm to test if p_i is prime. If p_i is not prime, retry from step 1. (Note: we may use either Adelman-Pomerance-Rumely [1] or Cohen-Lenstra [7] for this test.)

4. Return $((A_0, B_0), L_0, p_1), \dots, ((A_{i-1}, B_{i-1}), L_{i-1}, p_i)$.

Algorithm 7. Check-Prime($p_0, ((A_0, B_0), L_0, p_1), \dots, ((A_{i-1}, B_{i-1}), L_{i-1}, p_i)$):

1. Reject if $p_i > \max(2^{(\log p)^{C/\log\log\log p}}, 37)$.
2. Use a near-polynomial time deterministic algorithm to test if p_i is prime. If p_i is not prime, reject. (Note: again, we may use either Adelman-Pomerance-Rumely (1983) or Cohen-Lenstra (1984).)
3. For each j from 0 to $i-1$ inclusive, reject if any of the following assertions fail:
   (a) $2 \nmid p_j$
   (b) $3 \nmid p_j$
   (c) $(4A_j^3 + 27B_j^2, p_j) = 1$
   (d) $p_{j+1} > (\sqrt[4]{p_j} + 1)^2$
   (e) $L_j \neq \infty$
   (f) $p_{j+1} L_j = \infty$
4. Accept p_0 as prime.

4 Analysis

4.1 Overview

The overall approach of Goldwasser-Kilian is to reduce the question of the primality of p to the question of the primality of a smaller prime q, where $q \leq \frac{p}{2} + o(p)$. This results in a chain of primes $p_0, p_1, \dots, p_i$, where $p_0 = p$ is the original prime, and p_i is small enough to be verified deterministically in polynomial time. This chain is a certificate of primality for p and can be verified quickly in polynomial time. The following subsections with discuss each of the presented algorithms in greater detail.

4.2 Main reduction step

Generate-Curve generates a random elliptic curve $E_{A,B}(\mathbb{F}_p)$ with order $2q$, where q is a probable prime as determined by a probabilistic primality test. This is done by repeatedly sampling A and B randomly from $\mathbb{F}_p$ until the conditions hold. Note that we require the probabilistic primality test to err with an exponentially small probability (say, $1/p$, where p is the prime candidate). Nonetheless, the probabilistic test may be incorrect; we discuss later how to resolve such errors. Assume for now that q actually is prime.

Find-Point takes an output of Generate-Curve and finds a point $L \in E_{A,B}(\mathbb{F}_p)$ such that the order of L is q. This is easily done by randomly sampling L from $E_{A,B}(\mathbb{F}_p)$, since about half the points will have order q.

4.3 Primality proving

Prove-Prime repeatedly call the reduction algorithms to generate a chain of primes that forms the certificate of primality of $p = p_0$. The loop terminates when the final prime is small enough to be verified deterministically in time polynomial in $\log p$. Note that it is easily verified that such a constant C may always be found [9]. In the rare cases that the probabilistic test in Generate-Curve was incorrect, the main loop would not terminate, as the algorithm is trying to prove that a composite number is prime. The bound on the loop index $i < (\log p)^{\log \log p}$ in step 2(d) handles this case gracefully and does not increase asymptotic running time [9].

4.4 Certificate checking

Check-Prime constitutes a fast deterministic verification algorithm of a certificate generated by Prove-Prime. It is easy to see that if Check-Prime accepts a prime, then p_i is prime, and by Theorem 3, p_j prime implies p_{j-1} prime. So it suffices to show that all certificates generated by Prove-Prime satisfy the conditions required in the main loop.

The first two requirements that p_j not be divisible by 2 or 3 are ensured by the loop in Prove-Prime. Generate-Curve ensures that $(4A_j^3 + 27B_j^2, p_j) = 1$, hence satisfying requirement (c). Since $\#E_{A_j,B_j}(\mathbb{F}_{p_j}) \geq (\sqrt{p_j} - 1)^2$ by Hasse's Theorem, we have

$$\begin{aligned} p_{j+1} &\geq (\sqrt{p_j} - 1)^2/2 \\ &> (\sqrt[4]{p_j} + 1)^2 \end{aligned}$$

for all $p_j > 37$. But Prove-Prime ensures that $p_j > 37$, or otherwise it can be easily checked whether p_j is prime. This satisfies requirement (d). Finally, Find-Point ensures that $L_j \neq \infty$ and $p_{j+1}L_j = \infty$, satisfying the last two requirements. This proves that any certificate generated by Prove-Prime will be accepted by Check-Prime and constitutes a proof of primality of $p = p_0$.

4.5 Runtime

Since $p_{j+1} \approx p_j/2$, we expect $i = O(\log p)$ iterations of the main loop in Prove-Prime. For each of these iterations, we must consider the expected runtime of Generate-Curve and Find-Point.

The runtime of Generate-Curve is equal to the number of elliptic curves that must be tested before finding one of the right order, multiplied by the time needed to test for the order of each group. Let T_{p_j} be the number of curves that must be tested; we will defer discussion of the behavior of T_{p_j} until later. Schoof's algorithm runs in $O(\log^8 p_j)$ time [9], which dominates the $O(\log^4 p_j)$ runtime of the probabilistic primality test ($O(\log^3 p_j)$ time for each iteration of the test [7], multiplied by running $O(\log p_j)$ trials). Thus Generate-Curve runs in $T_{p_j} \cdot O(\log^8 p_j)$ time.

To analyze the runtime of Find-Point, we note that we need to choose an expected number of $\frac{2p}{\#E_{A,B}(\mathbb{F}_p)} \geq \frac{2p}{p-2\sqrt{p}+1} = O(1)$ values of x before we find a quadratic residue. Then since $E(\mathbb{F}_p) \cong \mathbb{Z}_2 \times \mathbb{Z}_q$, approximately half the points will be of order q, so we need to try 2 points on expectation. Verifying that $L \in E_{A,B}(\mathbb{F}_{p_j})$ has order q takes $O(\log^3 p_j)$ time, yielding a final expected runtime of $O(\log^3 p_j)$ time for Find-Point.

Putting all these terms together, Prove-Prime runs in $(\max T_{p_j})O(\log^9 p)$ time.

The runtime of Check-Prime is $O(\log^4 p)$, since each of $O(\log p)$ primes in the certificate chain may be verified in $O(\log^3 p)$.

The full analysis of the T_{p_j} term is outside the scope of this paper. We present the analysis under a widely-believed conjecture.

Conjecture 8. (Distribution of primes in short intervals):
Let $S(p) = \{q : p+1-\lfloor\sqrt{p}\rfloor \leq 2q \leq p+1+\lfloor\sqrt{p}\rfloor,\ q \text{ prime}\}$ be the set of primes in a short interval around $p/2$. Then $|S(p)| = O(\frac{\sqrt{p}}{\log^c p})$ for some constant c.

In other words, the distribution of primes in a short interval can be approximated by the asymptotic behavior of the distribution of primes.

Given Conjecture 8, we may apply Theorem 1 to get

Corollary 9. (Distribution of elliptic curve orders in short intervals):
Let $p > 5$ be a prime, and choose A, B randomly from $\mathbb{F}_p$. Let $S(p) \subseteq \{p+1-\lfloor\sqrt{p}\rfloor, \ldots, p+1+\lfloor\sqrt{p}\rfloor\}$ be defined as above to be the set of numbers in the interval that are twice a prime. Then

$$\Pr(\#E_{A,B}(\mathbb{F}_p) \in S(p)) = \Omega\left(\frac{1}{\log^{c+1} p}\right)$$

where c is the constant from Conjecture 8.

Proof. Letting $S = S(p)$ in Theorem 1, we get $\Pr(\#E_{A,B}(\mathbb{F}_p) \in S(p)) > \frac{k_1}{\log p} \cdot \frac{k_2\sqrt{p}}{\log^c p}\frac{1}{2\lfloor\sqrt{p}\rfloor+1} > \frac{k_3}{\log^{c+1} p} = \Omega\left(\frac{1}{\log^{c+1} p}\right)$ for some constants k_1, k_2, k_3. ■

Under this conjecture, we may invert the probability to get $\max T_{p_j} = T_p = O(\log^{c+1} p)$. This gives Prove-Prime an expected runtime of $O(\log^{c+10} p)$.

Without this conjecture, it is possible to show that Goldwasser-Kilian runs in expected time $O(\log^{12} p)$ for almost all inputs p [8].

5 Recent Advances

Since the publication of Goldwasser-Kilian, various improvements have been made on the ECPP. Atkin-Morain [5] decided to approach the problem from a different direction: instead of searching for a curve and then computing its order as in Goldwasser-Kilian, Atkin-Morain computes an order and searches for a curve of that order using complex multiplication. By doing so, Atkin-Morain avoids the expensive Schoof computation and runs much faster in practice, with a heuristic runtime of $O(\log^5 p)$ [5].

"Fast ECPP" improves on Atkin-Morain by using a faster method to compute many square roots and has achieved heuristic time $\tilde{O}(\log^4 p)$. Morain [12] points out that several steps in "fast ECPP" have time complexity $\tilde{O}(\log^3 p)$, so it seems that further advances in improving the efficiency would require major revisions. Cheng [6] has recently published a hybrid method of ECPP and AKS that also achieves heuristic time $\tilde{O}(\log^4 p)$.

6 Future Developments

Primality proving has come a long way from its early beginnings in ancient Greece. The establishment of polynomial-time proving algorithms is one of the major breakthroughs of modern mathematics. While specialized tests have contributed to the largest primes known (the Lucas-Lehmer test found a 12978189-digit Mersenne prime in 2008 [18]), generalized tests have allowed us to find large primes without a special form. ECPP algorithms have proven to be the fastest known generalized algorithms in practice, and the rise of parallel computation has given a substantial speed boost to these algorithms. For example, the largest prime established on a single-processor instance of Morain's ECPP implementation is 3508 digits, while a cluster-based implementation recently established a 26643-digit prime [13]. Combining strategies from ECPP and AKS may give insight into faster hybrid algorithms for primality proving.

Due to the sequential nature of the ECPP certificate generation and the lack of data parallelism in elliptic curve operations, GPUs seem ill-suited to handling the parallelism of elliptic curve primality proving. There is promise, however, in the use of GPUs to parallelize the probabilistic primality tests used by ECPP. A recent project has produced a GPU-based primality test that gives an order of magnitude stronger probable primes than previous tests [17]. Theoretically, stronger probable primes would decrease the number of retries encountered by ECPP, although as this number is already fairly low, GPU parallelism is unlikely to result in a marked empirical improvement in ECPP. As computing becomes increasingly parallel, research into developing data parallel techniques of primality proving becomes increasingly valuable.

References

[1] Adelman, L. M.; Pomerance, C.; Rumely, R. On distinguishing prime numbers from composite numbers. *Ann. of Math.* 117(1): 173-206, 1983. MR0683806 (84e:10008)

[2] Adelman, L. M.; Huang, M. A. Primality testing and Abelian varieties over finite fields. *Lecture Notes in Mathematics* vol. 1512: 142, 1992. MR0683806 (84e:10008)

[3] Agrawal, M.; Kayal, N.; Saxena, N. PRIMES is in P. *Ann. of Math.* 160(2), 781-793, 2004. MR2123939 (2006a:11170)

[4] Atkin, A. O. L. Manuscript, Lecture notes of a conferences, Boulder, Colorado, August 1986.

[5] Atkin, A. O. L.; Morain, F. Elliptic curves and primality proving. *Math. Comp.* 61(203): 29-68, 1993. MR1199989 (93m:11136)

[6] Cheng, Q. Primality proving via one round in ECPP and one iteration in AKS. *Advances in cryptology – CRYPTO 2003* in *Lecture Notes in Comput. Sci.* vol. 2729: 338-348, 2003. MR2093202 (2005e:68088)

[7] Cohen, H.; Lenstra, H. W., Jr. Primality testing and Jacobi sums. *Math. Comp.* 42(165): 197-330, 1984. MR0726006 (86g:11078)

[8] Goldwasser, S.; Kilian, J. Almost all primes can be quickly certified. *STOC '86 Proceedings of the 18th Annual ACM Symposium on Theory of Computing*: 316-329, 1986. doi:10.1145/12130.12162

[9] Goldwasser, S.; Kilian, J. Primality Testing Using Elliptic Curves. *Journal of the ACM* 46(4): 450-472, 1999. MR1812127 (2002e:11182)

[10] Lenstra, H. W., Jr. Factoring integers with elliptic curves. *Ann. of Math.* 126(3): 649-673, 1987. MR0916721 (89g:11125)

[11] Miller, G. L. Riemann's hypothesis and tests for primality. *J. Comput. System Sci.* 13(3): 300-317, 1976. MR0480295 (58 #470a)

[12] Morain, F. Implementing the Asymptotically Fast Version of the Elliptic Curve Primality Proving Algorithm. *Math. Comp.* 76(257): 493-505, 2007. MR2261033 (2007m:11167)

[13] Morain, F. Quelques nombres premiers prouvés par mes programmes (some primes proven by my programs). Available as `http://www.lix.polytechnique.fr/~morain/Primes/myprimes.html`

[14] Pratt, V. R. Every prime has a succinct certificate. *SIAM J. Comput.* 4(3): 214-220, 1975. MR0391574 (52 #12395)

[15] Rabin, M. O. Probabilistic algorithm for testing primality. *J. Number Theory* 12(1): 128-138, 1980. MR0566880 (81f:10003)

[16] Solovay, R.; Strassen, V. A fast Monte-Carlo test for primality. *SIAM J. Comput.* 6(1): 84-85, 1977. MR0429721 (55 #2732)

[17] Worley, S. Optimization of primality testing methods by GPU evolutionary search. *GPUs for Genetic and Evolutionary Computation*: 2009. Available as `http://www.gpgpgpu.com/gecco2009/6.pdf`

[18] Yates, S.; Caldwell, C. The largest known primes. Available as `http://primes.utm.edu/primes/lists/all.pdf`