J. Cryptology (2007) 20: 375–387
DOI: 10.1007/s00145-006-0406-9

Journal of
CRYPTOLOGY

Primality Proving via One Round in ECPP and One Iteration in AKS*

Qi Cheng
School of Computer Science, The University of Oklahoma,
Norman, OK 73019, U.S.A.
qcheng@cs.ou.edu

Communicated by Dan Boneh

Received 2 March 2004 and revised 23 June 2006
Online publication 7 February 2007

Abstract. In August 2002, Agrawal, Kayal and Saxena announced the first deterministic and polynomial-time primality-testing algorithm. For an input n, the Agarwal–Kayal–Saxena (AKS) algorithm runs in time $\tilde{O}(\log^{7.5} n)$ (heuristic time $\tilde{O}(\log^6 n)$). Verification takes roughly the same amount of time. On the other hand, the Elliptic Curve Primality Proving algorithm (ECPP) runs in random heuristic time $\tilde{O}(\log^6 n)$ (some variant has heuristic time complexity $\tilde{O}(\log^4 n)$) and generates certificates which can be easily verified. However, it is hard to analyze the provable time complexity of ECPP even for a small portion of primes. More recently, Berrizbeitia gave a variant of the AKS algorithm, in which some primes (of density $O(1/\log^2 n)$) cost much less time to prove than a general prime does. Building on these celebrated results, this paper explores the possibility of designing a randomized primality-proving algorithm based on the AKS algorithm. We first generalize Berrizbeitia's algorithm to one which has higher density ($\Omega(1/\log\log n)$) of primes whose primality can be proved in time complexity $\tilde{O}(\log^4 n)$. For a general prime, one round of ECPP is deployed to reduce its primality proof to the proof of a random easily proved prime, thus we achieve heuristic time complexity $\tilde{O}(\log^4 n)$ for all primes.

Key words. Computational number theory, Primality testing, Elliptic curve primality proving, Agrawal–Kayal–Saxena primality-testing algorithm, Algorithm analysis.

1. Introduction

Testing whether a number is prime or not is one of the fundamental problems in computational number theory. It has wide applications in computer science, especially in cryptography. After tremendous efforts invested by researchers in hundreds of years, it

* The preliminary version of this paper appeared in the *Proceedings of Crypto* 2003, Lecture Notes in Computer Science 2729, Springer-Verlag. This research was partially supported by NSF Career Award CCR-0237845.

was finally proved by Agrawal, Kayal and Saxena [3] that the set of primes is in the complexity class **P**. For a given integer n, the Agarwal–Kayal–Saxena (AKS) algorithm runs in time $\tilde{O}(\log^{7.5} n)$, while the best previously known deterministic algorithm has subexponential complexity [2]. Under a conjecture concerning the density of the Sophie–Germain primes, the AKS algorithm should give out an answer in time $\tilde{O}(\log^6 n)$.

Notation: In this paper we use "ln" for logarithm base e and "log" for logarithm base 2. We write $r^\alpha || n$, if $r^\alpha | n$ but $r^{\alpha+1} \nmid n$. By $\tilde{O}(f(n))$, we mean $O(f(n)\log^c(f(n)))$ for some constant c.

The AKS algorithm is based on the derandomization of a polynomial identity testing. It includes many iterations of polynomial modular exponentiation. To test the primality of an integer n, the algorithm first searches for a suitable r, which is provably $O(\log^3 n)$ or heuristically $O(\log^2 n)$. Then the algorithm checks for l from 1 to $L = \lceil 2\sqrt{r}\log n \rceil$, whether

$$(x + l)^n \equiv x^n + l \pmod{n, x^r - 1}. \tag{1}$$

The algorithm declares that n is a prime if all the congruences hold. The computing of $(x + l)^n \pmod{n, x^r - 1}$ takes time $\tilde{O}(r\log^2 n)$ if we use the fast multiplication. The total time complexity is thus $\tilde{O}(rL\log^2 n)$.

While the AKS algorithm is a great accomplishment in the theory, the current version is very slow. Unless its time complexity can be dramatically improved, it cannot replace random primality-testing algorithms with better efficiency. Even for a cryptosystem requiring a prime with absolute certainty, an efficient random algorithm is sufficient, as long as the algorithm can generate a certificate of primality, which in deterministic time convinces a verifier who does not believe any number theory conjectures. A primality-testing algorithm that generates a certificate of primality is sometimes called *a primality-proving algorithm*. Similarly a primality-testing algorithm that generates a certificate of compositeness is sometimes called *a compositeness-proving algorithm*. If a compositeness-proving algorithm fails to prove the compositeness of a number n after a certain number of tries, then n is called a *probable prime*. Very efficient random compositeness-proving algorithms have long been known. Curiously, primality-proving algorithms [5], [1] lag far behind compositeness-proving algorithms in terms of efficiency and simplicity.

Recently, Berrizbeitia [8] proposed a brilliant modification to the original AKS algorithm. He used the polynomial $x^{2^s} - a$ instead of $x^r - 1$ in (1), where $2^s \approx \log^2 n$. Among others, he was able to prove the following proposition:

Proposition 1. *Given an integer $n \equiv 1 \pmod 4$, denote $s = \lceil 2\log\log n \rceil$. Assume that $2^k || n - 1$ and $k \geq s$. If there exists an integer a, such that $(a/n) = -1$ and $a^{(n-1)/2} \equiv -1 \pmod n$, then*

$$(1 + x)^n \equiv 1 + x^n \pmod{n, x^{2^s} - a}$$

iff n is a power of a prime.

Unlike the AKS algorithm, where each prime costs roughly the same, there are "easily proved primes" in Berrizbeitia's algorithm, namely, the primes p where $p-1$ has a factor of a power of 2 larger than $\log^2 n$. For those primes, one iteration of polynomial modular

exponentiation, which runs in time $\tilde{O}(\log^4 n)$, establishes the primality of n, provided that a suitable a exists. In fact, a can be found easily if n is indeed a prime and randomness is allowed in the algorithm. It serves as a prime certificate for n.

Definition 1. In this paper, for a primality-proving algorithm, we call a prime p *easily proved*, if the algorithm runs in expected time $\tilde{O}(\log^4 p)$ on p.

What is the density of the easily proved primes in Berrizbeitia's algorithm? Let $\pi(x)$ denote the number of primes less than or equal to x. The number of primes of form $2^s x + 1$ less than b is about $\pi(b)/\varphi(2^s)$. Hence heuristically for a random prime p, $p-1$ has a factor $2^s \approx \log^2 p$ with probability no larger than $2/\log^2 p$. In the other words, the easily proved primes have density at most $2/\log^2 p$ around p in his algorithm.

1.1. *Increasing the Density of Easily Proved Primes*

We prove the following theorem in Section 5, which can be regarded as a generalization of Proposition 1.

Theorem 1 (Main). *Given a number n which is not a power of an integer, suppose that there exists a prime r, $r^\alpha || n-1 (\alpha \geq 1)$ and $r \geq \log^2 n$. In addition, suppose that there exists a number $1 < a < n$, such that $a^{r^\alpha} \equiv 1 \pmod n$, $\gcd(a^{r^{\alpha-1}} - 1, n) = 1$ and*

$$(1+x)^n = 1 + x^n \pmod{n, x^r - a},$$

then n is a prime.

The number a can be found easily if n is a prime and randomness is allowed. It, in addition to r, serves as a prime certificate for n. Based on this theorem, we propose a random algorithm which establishes the primality of p in time $\tilde{O}(\log^4 p)$ if $p-1$ contains a prime factor between $a \log^2 p$ and $b \log^2 p$ for some constants a and b.

Definition 2. We call a positive integer n (a,b)-*good*, if $n-1$ has a prime factor p such that $a \log^2 n \leq p \leq b \log^2 n$. We call n good, if it is (1,2)-good.

What is the density of (a,b)-good primes? Clearly the density should be higher than the density of easily proved primes in Berrizbeitia's algorithm. We will prove the following theorem:

Theorem 2. *Let a, b be two constants. Let $M_{a,b}(x)$ denote the number of primes $p \leq x$ which are (a,b)-good, then*

$$M_{a,b}(x) = \frac{\ln(b/a)}{2 \ln\ln x} \pi(x) + O\left(\frac{\pi(x)}{(\ln\ln x)^2}\right).$$

1.2. *Algorithm for the General Primes*

For general primes, we apply the idea in the Elliptic Curve Primality Proving algorithm (ECPP). ECPP was proposed by Goldwasser and Kilian [9] and Atkin [4] and imple-

mented by Atkin and Morain [5] and Kaltofen et al. [10]. In practice, ECPP performs much better than the current version of AKS. It has been used to prove primality of numbers up to thousands of decimal digits [12].

In ECPP, if we want to prove that an integer n is a prime, we reduce the problem to the proof of primality of a smaller number (less than $n/2$). To achieve this, we try to find an elliptic curve with $\omega n'$ points over $\mathbf{Z}/n\mathbf{Z}$, where ω is completely factored and n' is a probable prime greater than $(\sqrt[4]{n}+1)^2$. Once we have such a curve and a point on the curve with order n', the primality of n' implies the primality of n. Since point counting on elliptic curves is expensive, we usually use the elliptic curves with complex multiplications of small discriminants. Nonetheless, it is plausible to assume that the order of the curve has the desired form with the same probability as a random integer does. ECPP needs $O(\log n)$ rounds of reductions to eventually reduce the problem to a primality proof of a very small prime, say, less than 50000. As observed in [11], one round of reduction takes heuristic time $\tilde{O}(\log^5 n)$, or $\tilde{O}(\log^4 n)$ if we use the fast multiplication. To analyze the time complexity, it is assumed that the number of primes between $n-2\sqrt{n}+1$ and $n+2\sqrt{n}+1$ is greater than $\sqrt{n}/\log^2 n$, and the number of points on an elliptic curve with small discriminant complex multiplication behaves like a random number in this range. We refer to the assumption as the ECPP heuristics. For the ECPP algorithm, rigorous proof of the time complexity seems out of reach, as it involves the study of the prime distribution in a short interval. Due to the nature of using special curves, it is not feasible to prove the time complexity for even a small fraction of the primes.

Remark. There is a faster variant of elliptic curve primality-proving algorithm, proposed by Shallit and reported in [11], which runs in heuristic time $\tilde{O}(\log^4 n)$. However, it has not been tested extensively. It appears that the variant is faster than the ECPP only for very large numbers.

Our algorithm consists of two stages. In the first stage, for a general probable prime n, we use one round of ECPP to reduce its proof of primality to a good probable prime n' close to n. For convenience, we require that $n-2\sqrt{n}+1 \le n' \le n+2\sqrt{n}+1$ (see Section 6 for implementation issues). Note that up to a constant factor, the time complexity of one round reduction in ECPP is equivalent to the time complexity of finding a curve with a prime order. We conjecture that Theorem 2 is true in a shorter interval, for example, between $n-2\sqrt{n}+1$ and $n+2\sqrt{n}+1$.

Conjecture 1. *There exists an absolute constant* λ, *such that for any sufficiently large integer* n,

$$\frac{\textit{Number of } (1,2)\textit{-good primes between } n-2\sqrt{n}+1 \textit{ and } n+2\sqrt{n}+1}{\textit{Number of primes between } n-2\sqrt{n}+1 \textit{ and } n+2\sqrt{n}+1} > \frac{\lambda}{\ln\ln n}.$$

We are unable to prove this conjecture, but we present in the paper some numerical evidence. According to the above conjecture and the ECPP heuristics, *the extra condition on* n' (*that* n' *should be good*) *will increase the time complexity of finding* n'

merely by a factor of $O(\log\log n)$. Therefore for an arbitrary prime, without significant increase of time complexity, we reduce its primality proving to the proof of a good prime. In the second stage we find a primality certificate for n'. To do this, we search for a which satisfies the conditions in the main theorem, and we compute the polynomial modular exponentiation. The total expected running time of the first and second stages becomes $\tilde{O}(\log^4 n)$. However, because of the reasons mentioned before, it seems difficult to obtain the rigorous time complexity for the first step. Put it all together, and we now have a general-purpose prime-proving algorithm with the following properties:

1. It proves primality of a significant part of primes in time $\tilde{O}(\log^4 n)$, without assuming any conjecture. For those primes, the ECPP subroutine is not needed.
2. It runs very fast ($\tilde{O}(\log^4 n)$) on every prime, assuming reasonable conjectures.
3. The certificate, which consists of an elliptic curve—a point on the curve with order n', n', r and a—is very short. It consists of only $O(\log n)$ bits as opposed to $O(\log^2 n)$ bits in ECPP.
4. A verifier can be convinced in deterministic time $\tilde{O}(\log^4 n)$. In fact, the most time-consuming part in the verification is the computation of one polynomial modular exponentiation.

This paper is organized as follows: In Section 2 we review the propositions used by AKS and ECPP to prove primality. In Section 3 we describe our algorithm and present the time-complexity analysis. In Section 4 we prove Theorem 2 and present some numerical evidence for Conjecture 1. The main theorem is proved in Section 5. We conclude this paper with some discussions on the implementation of the algorithm.

2. Proving Primality in AKS and ECPP

The ECPP algorithm depends on rounds of reductions of the proof of primality of a prime to the proof of primality of a smaller prime. The most remarkable feature of ECPP is that once the certificate is constructed, a verifier who does not believe any conjectures can be convinced in time $\tilde{O}(\log^3 n)$ if the fast multiplication is used. It is based on the following proposition [5]:

Proposition 2. *Let N be an integer prime to* 6, *and let E be an elliptic curve over $\mathbf{Z}/N\mathbf{Z}$, together with a point P on E and two integers m and s with $s|m$. Denote the infinite point on E by O. For each prime divisor q of s, denote $(m/q)P$ by $(x_q : y_q : z_q)$. Assume that $mP = O$ and $\gcd(z_q, N) = 1$ for all q. If $s > (\sqrt[4]{N} + 1)^2$, then N is a prime.*

The certificate for N in ECPP consists of the curve E, the point P, m, s and the certificate of primality of s. Usually the ECPP uses elliptic curves with complex multiplications of small discriminants. For implementation details, see [5].

The AKS algorithm, on the other hand, proves a number is a prime through the following proposition:

Proposition 3. *Let n be a positive integer. Let q and r be prime numbers. Let S be a finite set of integers. Assume*

1. *that q divides $r-1$;*
2. *that $n^{(r-1)/q} \not\equiv 0, 1 \pmod{r}$;*
3. *that $\gcd(n, b-b') = 1$ for all the distinct $b, b' \in S$;*
4. *that $\binom{q+|S|-1}{|S|} \geq n^{2\lfloor\sqrt{r}\rfloor}$; and*
5. *that $(x+b)^n \equiv x^n + b \pmod{x^r - 1, n}$ for all $b \in S$.*

Then n is a power of a prime.

3. Description and Time-Complexity Analysis of Our Algorithm

Now we are ready to describe our algorithm.

Input: a positive integer n.
Output: a certificate of primality of n, or "composite."

1. If n is a power of an integer, return "composite."
2. Run a random compositeness-proving algorithm, for example, the Rabin–Miller testing [6, page 282], on n. If a proof of compositeness is found, output the proof, return "composite" and exit.
3. If $n-1$ contains a prime factor between $\log^2 n$ and $2\log^2 n$, skip this step. Otherwise, call the ECPP reduction procedure to find an elliptic curve over $\mathbf{Z}/n\mathbf{Z}$ with n' points, where n' is a probable prime and n' is good. Set $n = n'$.
4. Let r be the prime factor of $n-1$ satisfying $\log^2 n \leq r \leq 2\log^2 n$. Let α be the integer such that $r^\alpha || n-1$.
5. Randomly select an integer $1 < b < n$. If $b^{n-1} \neq 1 \pmod{n}$, exit.
6. Let $a = b^{(n-1)/r^\alpha} \pmod{n}$. If $a = 1$, or $a^{r^{\alpha-1}} = 1$, go back to step 5.
7. If $\gcd(a^{r^{\alpha-1}} - 1, n) \neq 1$, exit.
8. If $(1+x)^n \neq 1 + x^n \pmod{n, x^r - a}$, exit.
9. Output a. If in step 3, the ECPP procedure is invoked, find a point on the curve, output the curve, the point and the order of the curve. Return "prime."

On any input integer n, this algorithm will either output "composite" with a compositeness proof, or "prime" with a primality proof, or nothing. It outputs nothing only when the probable prime n in Step 4 is actually composite. The probability depends on the number of iterations in the compositeness-proving algorithm. If we use $\log\log n$ iterations, it is $O(1/\log n)$.

Now we analyze its time complexity. Testing whether a number n is good or not can be done in time $\tilde{O}(\log^3 n)$. Step 3 takes time $\tilde{O}(\log^4 n)$ if the ECPP heuristics is true, Conjecture 1 is true and the fast multiplication algorithm is used. If n is indeed a prime, then the probability of going back in step 6 is at most $1/r$. Step 7 takes time at most $\tilde{O}(\log^2 n)$. Step 8 takes time $\tilde{O}(\log^4 n)$, since $r \leq 2\log^2 n$. Hence the heuristic expected running time of our algorithm is $\tilde{O}(\log^4 n)$. Obviously the verification algorithm takes deterministic time $\tilde{O}(\log^4 n)$.

4. Density of Good Numbers

Let $\pi(x, k, a)$ denote the number of primes $p \le x$ with $p \equiv a \pmod{k}$. For $k \le \ln^5 x$ and $\gcd(a, k) = 1$, by the Siegel–Walfisz theorem, we have

$$\pi(x, k, a) = \frac{\pi(x)}{\varphi(k)} + O\left(\frac{\pi(x)}{\varphi(k)\ln^2 x}\right).$$

First we prove a lemma.

Lemma 1. *Assume that a and b are functions of x, $0 < a < b$, $a = \Theta(1)$ and $b = \Theta(1)$. Let $N(x, a, b)$ denote the number of primes $p \le x$ such that $p - 1$ has a prime factor in $(a \ln^2 x, b \ln^2 x)$, then we have*

$$N(x, a, b) = \frac{\ln(b/a)}{2\ln\ln x}\pi(x) + O\left(\frac{\pi(x)}{(\ln\ln x)^2}\right).$$

Proof. Let q, q_1, q_2 run over the primes between $a \ln^2 x$ and $b \ln^2 x$. By the inclusion–exclusion law, we have

$$\sum_q \pi(x, q, 1) - \sum_{q_1 < q_2} \pi(x, q_1 q_2, 1) \le N(x, a, b) \le \sum_q \pi(x, q, 1).$$

From the Siegel–Walfisz theorem, we obtain

$$\sum_q \pi(x, q, 1) = \pi(x) \sum_q \frac{1}{q-1} + O\left(\frac{\pi(x)}{\ln^2 x} \sum_q \frac{1}{q-1}\right)$$

and we also have

$$\sum_q \frac{1}{q-1} = \sum_q \frac{1}{q} + \sum_q \frac{1}{q(q-1)}.$$

Since

$$\begin{aligned}
\sum_q \frac{1}{q} &= \sum_{q \le b\ln^2 x} \frac{1}{q} - \sum_{q < a\ln^2 x} \frac{1}{q} \\
&= \ln\ln(b\ln^2 x) - \ln\ln(a\ln^2 x) + O\left(\frac{1}{(\ln\ln x)^2}\right) \\
&= \ln\frac{2\ln\ln x + \ln b}{2\ln\ln x + \ln a} + O\left(\frac{1}{(\ln\ln x)^2}\right) \\
&= \ln\left(1 + \frac{\ln(b/a)}{2\ln\ln x + \ln a}\right) + O\left(\frac{1}{(\ln\ln x)^2}\right) \\
&= \frac{\ln(b/a)}{2\ln\ln x + \ln a} + O\left(\frac{1}{(\ln\ln x)^2}\right) \\
&= \frac{\ln(b/a)}{2\ln\ln x} + O\left(\frac{1}{(\ln\ln x)^2}\right)
\end{aligned}$$

and

$$\sum_q \frac{1}{q(q-1)} \le \sum_{a\ln^2 x \le n \le b\ln^2 x} \frac{1}{n(n-1)} = \frac{1}{a\ln^2 x - 1} - \frac{1}{b\ln^2 x} = O\left(\frac{1}{\ln^2 x}\right),$$

we have

$$\sum_q \frac{1}{q-1} = \frac{\ln(b/a)}{2\ln\ln x} + O\left(\frac{1}{(\ln\ln x)^2}\right).$$

Here we use the fact that $b/a = O(1)$ and $a = O(1)$. We also apply the Mertens theorem:

$$\sum_{p\le y} \frac{1}{p} = \ln\ln y + B + O\left(\frac{1}{(\ln y)^2}\right).$$

Note that we use a better error term than the one in the standard form. It remains to show that

$$\sum_{q_1<q_2} \pi(x, q_1q_2, 1) = O\left(\frac{\pi(x)}{(\ln\ln x)^2}\right).$$

This is true since

$$\sum_{q_1<q_2} \pi(x, q_1q_2, 1) = \pi(x) \sum_{q_1<q_2} \frac{1}{(q_1-1)(q_2-1)} + O\left(\frac{\pi(x)}{\ln^2 x}\right)$$

and

$$\sum_{q_1<q_2} \frac{1}{(q_1-1)(q_2-1)} < \left(\sum_q \frac{1}{q-1}\right)^2 = O\left(\frac{1}{(\ln\ln x)^2}\right).$$

It concludes the proof of the lemma. □

Now we are ready to prove Theorem 2. Note that if $x/\ln x < p \le x$, then $\ln^2 x - \ln x \ln\ln x < \ln^2 p \le \ln^2 x$. We have

$$M_{a,b}(x) \le N(x, a, b)$$

and

$$\begin{aligned} M_{a,b}(x) &> N\left(x, a - \frac{\ln\ln x}{\ln x}, b - \frac{\ln\ln x}{\ln x}\right) - \pi\left(\frac{x}{\ln x}\right) \\ &= N\left(x, a - \frac{\ln\ln x}{\ln x}, b - \frac{\ln\ln x}{\ln x}\right) + O\left(\frac{\pi(x)}{\ln x}\right). \end{aligned}$$

Hence by the above lemma,

$$M_{a,b}(x) = \frac{\pi(x)\ln(b/a)}{2\ln\ln x} + O\left(\frac{\pi(x)}{(\ln\ln x)^2}\right).$$

See Table 1 for numerical data concerning the density of good primes around 2^{500}. For comparison, note that

$$\frac{\ln(2)}{2\ln\ln(2^{500})} = 0.0592626.$$

Table 1. Number of (1, 2)-good primes around 2^{500}.

From	To	Number of primes	Number of (1, 2)-good primes	Ratio (%)
$2^{500}+0$	$2^{500}+200000$	576	35	6.07
$2^{500}+200000$	$2^{500}+400000$	558	38	6.81
$2^{500}+400000$	$2^{500}+600000$	539	30	5.56
$2^{500}+600000$	$2^{500}+800000$	568	23	4.05
$2^{500}+800000$	$2^{500}+1000000$	611	39	6.38
$2^{500}+1000000$	$2^{500}+1200000$	566	26	4.59
$2^{500}+1200000$	$2^{500}+1400000$	566	38	6.71
$2^{500}+1400000$	$2^{500}+1600000$	526	27	5.13
$2^{500}+1600000$	$2^{500}+1800000$	580	26	4.48
$2^{500}+1800000$	$2^{500}+2000000$	563	20	3.55
$2^{500}+2000000$	$2^{500}+2200000$	562	22	3.91
$2^{500}+2200000$	$2^{500}+2400000$	561	21	3.74
$2^{500}+2400000$	$2^{500}+2600000$	609	34	5.58
$2^{500}+2600000$	$2^{500}+2800000$	601	28	4.66
$2^{500}+2800000$	$2^{500}+3000000$	603	33	5.47
$2^{500}+3000000$	$2^{500}+3200000$	579	37	6.39
$2^{500}+3200000$	$2^{500}+3400000$	576	31	5.38
$2^{500}+3400000$	$2^{500}+3600000$	604	35	5.79
$2^{500}+3600000$	$2^{500}+3800000$	612	40	6.53
$2^{500}+3800000$	$2^{500}+4000000$	588	29	4.93
$2^{500}+4000000$	$2^{500}+4200000$	574	33	5.75
$2^{500}+4200000$	$2^{500}+4400000$	609	27	4.43
$2^{500}+4400000$	$2^{500}+4600000$	549	35	6.37
$2^{500}+4600000$	$2^{500}+4800000$	561	30	5.34
$2^{500}+4800000$	$2^{500}+5000000$	545	29	5.32
$2^{500}+5000000$	$2^{500}+5200000$	590	20	3.39
$2^{500}+5200000$	$2^{500}+5400000$	557	27	4.84
$2^{500}+5400000$	$2^{500}+5600000$	591	28	4.73
$2^{500}+5600000$	$2^{500}+5800000$	517	33	6.38
$2^{500}+5800000$	$2^{500}+6000000$	566	18	3.18
$2^{500}+6000000$	$2^{500}+6200000$	575	30	5.21
$2^{500}+6200000$	$2^{500}+6400000$	573	26	4.53
$2^{500}+6400000$	$2^{500}+6600000$	558	36	6.45
$2^{500}+6600000$	$2^{500}+6800000$	574	32	5.57
$2^{500}+6800000$	$2^{500}+7000000$	594	22	3.70
$2^{500}+7000000$	$2^{500}+7200000$	596	31	5.20
$2^{500}+7200000$	$2^{500}+7400000$	567	26	4.58
$2^{500}+7400000$	$2^{500}+7600000$	619	28	4.52
$2^{500}+7600000$	$2^{500}+7800000$	565	25	4.42
$2^{500}+7800000$	$2^{500}+8000000$	561	25	4.45
$2^{500}+8000000$	$2^{500}+8200000$	570	26	4.56

5. Proof of the Main Theorem

In this section we prove the main theorem. It is built on a series of lemmas. Some of them are straightforward generalizations of the lemmas in Berrizbeitia's paper [8]. We include slightly different proofs of those lemmas, though, for completeness.

Lemma 2. *Let r, p be primes, $r|p-1$. If $a \in \mathbf{F}_p$ is not an rth power of any element in $\mathbf{F}_p$, then $x^r - a$ is irreducible over $\mathbf{F}_p$.*

Proof. Let θ be one of the roots of $x^r - a = 0$. Certainly $[\mathbf{F}_p(\theta)\colon \mathbf{F}_p] > 1$. Let $\xi \in \mathbf{F}_p$ be one of the rth primitive roots of unity.

$$x^r - a = x^r - \theta^r = \prod_{0 \le i \le r-1} (x - \xi^i \theta).$$

Let $[\mathbf{F}_p(\theta)\colon \mathbf{F}_p] = r'$. Then for all i, $[\mathbf{F}_p(\xi^i\theta)\colon \mathbf{F}_p] = r'$. Hence over $\mathbf{F}_p$, $x^r - a$ is factored into polynomials of degree r' only. Since r is a prime, this is impossible, unless that $r' = r$. □

Lemma 3. *Let $n > 2$ be an integer. Let r be a prime and $r^\alpha || n-1$. Suppose that there exists an integer $1 < a < n$ such that*

1. $a^{r^\alpha} \equiv 1 \pmod{n}$;
2. $\gcd(a^{r^{\alpha-1}} - 1, n) = 1$;

then there must exist a prime factor p of n, such that $r^\alpha || p-1$ and a is not an rth power of any element in $\mathbf{F}_p$.

Proof. For any prime factor q of n, $a^{r^\alpha} \equiv 1 \pmod{q}$ and $a^{r^{\alpha-1}} \not\equiv 1 \pmod{q}$, so $r^\alpha | q-1$. If $r^{\alpha+1} | q-1$ for all the prime factors, then $r^{\alpha+1} | n-1$, a contradiction. Hence there exists a prime factor p, such that $r^\alpha || p-1$. Let g be a generator in $\mathbf{F}_p^*$. If $a = g^t$ in $\mathbf{F}_p$, then $p-1 | tr^\alpha$, and $p-1 \nmid tr^{\alpha-1}$. Hence $r \nmid t$. □

In the following text, we assume that n is an integer, $n = p^l d$ where p is a prime and $\gcd(p, d) = 1$. Assume r is a prime and $r|p-1$. Let $x^r - a$ be an irreducible polynomial in $\mathbf{F}_p$. Let θ be one of the roots of $x^r - a$. For any element in the field $\mathbf{F}_p(\theta)$, we can find a unique polynomial $f \in \mathbf{F}_p[x]$ of degree less than r such that the element can be represented by $f(\theta)$. Define $\sigma_m\colon \mathbf{F}_p(\theta) \to \mathbf{F}_p(\theta)$ as $\sigma_m(f(\theta)) = f(\theta^m)$.

Lemma 4. *We have that $a^m = a$ in $\mathbf{F}_p$ iff $\sigma_m \in \mathrm{Gal}(\mathbf{F}_p(\theta)/\mathbf{F}_p)$.*

Proof. ($\Leftarrow$) Since $\sigma_m \in \mathrm{Gal}(\mathbf{F}_p(\theta)/\mathbf{F}_p)$, θ^m must be a root of $x^r - a$. Hence $a = (\theta^m)^r = a^m$ in $\mathbf{F}_p$.

($\Rightarrow$) For any two elements $\alpha, \beta \in \mathbf{F}_p(\theta)$, we need to prove that $\sigma_m(\alpha + \beta) = \sigma_m(\alpha) + \sigma_m(\beta)$ and $\sigma_m(\alpha\beta) = \sigma_m(\alpha)\sigma_m(\beta)$. The first one is trivial from the definition of σ_m. Suppose that α is represented by $f_\alpha(\theta)$ and β is represented by $f_\beta(\theta)$ where $f_\alpha(x), f_\beta(x) \in \mathbf{F}_p[x]$ has degree at most $r-1$. If $\deg(f_\alpha(x) f_\beta(x)) \le r-1$, it is easy to see that $\sigma_m(\alpha\beta) = \sigma_m(\alpha)\sigma_m(\beta)$. Now assume that $\deg(f_\alpha(x) f_\beta(x)) \ge r$. Then $f_\alpha(x) f_\beta(x) = h(x) + (x^r - a) p(x)$ where $h(x), p(x) \in \mathbf{F}_p[x]$ and $\deg(h(x)) < r$. Then $\sigma_m(\alpha\beta) = \sigma_m(h(\theta)) = h(\theta^m) = h(\theta^m) + (a^m - a) p(\theta^m) = h(\theta^m) + (\theta^{mr} - a) p(\theta^m) = f_\alpha(\theta^m) f_\beta(\theta^m) = \sigma_m(\alpha)\sigma_m(\beta)$.

This shows that σ_m is a homomorphism. To complete the proof, we need to show that it is also one-to-one. This is obvious since $\mathbf{F}_p(\theta^m) = \mathbf{F}_p(\theta)$. □

Define $G_m = \{f(\theta) \in \mathbf{F}_p(\theta)^* | f(\theta^m) = f(\theta)^m\}$. It can be verified that G_m is a group when σ_m is in $\mathrm{Gal}(\mathbf{F}_p(\theta)/\mathbf{F}_p)$.

Lemma 5. *Suppose* $\sigma_n \in \mathrm{Gal}(\mathbf{F}_p(\theta)/\mathbf{F}_p)$. *Then for any* i, $j \geq 0$, $\sigma_{d^i p^j} \in \mathrm{Gal}(\mathbf{F}_p(\theta)/\mathbf{F}_p)$ *and* $G_n \subseteq G_{d^i p^j}$.

Proof. Notice that the map $x \to x^{p^l}$ is a one-to-one map in $\mathbf{F}_p(\theta)$. Since $\sigma_n \in \mathrm{Gal}(\mathbf{F}_p(\theta)/\mathbf{F}_p)$, we have $a^n = a$, according to the above lemma. This implies that $(a^d)^{p^l} = a$, hence $a^d = a$ and $a^{d^i p^j} = a$. We have $\sigma_{d^i p^j} \in \mathrm{Gal}(\mathbf{F}_p(\theta)/\mathbf{F}_p)$.

Let $f(\theta) \in G_n$. Thus $f(\theta^n) = f(\theta)^n$, this implies $f(\theta^{p^l d}) = f(\theta)^{p^l d} = f(\theta^{p^l})^d$. So θ^{p^l} is a solution of $f(x^d) = f(x)^d$. Since it is one of the conjugates of θ, θ must be a solution as well. This proves that $f(\theta^d) = f(\theta)^d$. Similarly since θ^d is also one of the conjugates of θ, as $\sigma_d \in \mathrm{Gal}(\mathbf{F}_p(\theta)/\mathbf{F}_p)$, we have $f(\theta^{d^2}) = f(\theta^d)^d = f(\theta)^{d^2}$. By induction, $f(\theta^{d^i}) = f(\theta)^{d^i}$ for $i \geq 0$. Hence $f(\theta^{d^i p^j}) = f(\theta^{d^i})^{p^j} = f(\theta)^{d^i p^j}$. This implies that $f(\theta) \in G_{d^i p^j}$. □

Lemma 6. *If* $\sigma_{m_1}, \sigma_{m_2} \in \mathrm{Gal}(\mathbf{F}_p(\theta)/\mathbf{F}_p)$ *and* $\sigma_{m_1} = \sigma_{m_2}$, *then* $|G_{m_1} \cap G_{m_2}|$ *divides* $m_1 - m_2$.

This lemma is straightforward from the fact that for any $g \in G_{m_1} \cap G_{m_2}$, $g^{m_1 - m_2} = 1$.

Lemma 7. *Let* $A = a^{r^{\alpha-1}}$. *If* $(1+\theta) \in G_n$, *so is* $1 + A^i\theta$ *for any* $i = 1, 2, 3, \ldots, r-1$, *and* $|G_n| \geq 2^r$.

Proof. If $(1+\theta) \in G_n$, this means that $(1+\theta)^n = 1 + \theta^n$. It implies that $(1+\theta')^n = 1 + \theta'^n$ for any conjugate θ' of θ. Since A is a primitive root of unity in $\mathbf{F}_p$, hence $A^i\theta$ are conjugates of θ. We have $(1 + A^i\theta)^n = 1 + (A^i\theta)^n = 1 + (A^n)^i\theta^n$ and we know that $A^n = A$. This proves that $1 + A^i\theta \in G_n$. The group G_n contains all the elements in the set

$$\left\{ \prod_{i=0}^{r-1} (1 + A^i\theta)^{\varepsilon_i} \Big| \sum_{i=0}^{r-1} \varepsilon_i < r \right\}.$$

By counting we have $|G_n| = \binom{2r+1}{r} \geq 2^r$. □

Finally we are ready to give the proof of the main theorem (Theorem 1) of this paper.

Proof. Since $|\mathrm{Gal}(\mathbf{F}_p(\theta)/\mathbf{F}_p)| = r$, hence there exist two different pairs (i_1, j_1) and (i_2, j_2) with $0 \leq i_1, j_1, i_2, j_2 \leq \lfloor\sqrt{r}\rfloor$, such that $\sigma_{d^{i_1} p^{j_1}} = \sigma_{d^{i_2} p^{j_2}}$. According to Lemma 5, $G_n \subseteq G_{d^{i_1} p^{j_1}}$, $G_n \subseteq G_{d^{i_2} p^{j_2}}$, this implies that $G_n \subseteq G_{d^{i_1} p^{j_1}} \cap G_{d^{i_2} p^{j_2}}$. Therefore $|G_n|$ divides $d^{i_1} p^{j_1} - d^{i_2} p^{j_2}$, but $d^{i_1} p^{j_1} - d^{i_2} p^{j_2} < n^{\lfloor\sqrt{r}\rfloor} \leq 2^{\sqrt{r}\log n} \leq 2^r$. hence $d^{i_1} p^{j_1} - d^{i_2} p^{j_2} = 0$, which in turn implies that n is a power of p. □

6. Implementation and Conclusion

In this paper we propose a random primality-proving algorithm which runs in heuristic time $\tilde{O}(\log^4 n)$. It generates a certificate of primality of length $O(\log n)$ which can be verified in deterministic time $\tilde{O}(\log^4 n)$. For $\pi(x)/\ln\ln x$ number of primes less than x, the algorithm runs in provable time $\tilde{O}(\log^4 n)$. Our results again illustrate that a polynomial-time algorithm usually has a practical version, with exponent 4 or less.

When it comes to implementing the algorithm, space is a bigger issue than time. Assume that n has 1000 bits when written in binary, which is in the range of practical interests. To compute $(1+x)^n \pmod{n, x^r - a}$, we have to store intermediate polynomials of size at least 2^{30} bits, or 128M bytes. Even taking the various constant improvement on r into consideration, it is still impossible to put a polynomial of degree r into the cache of a processor in a desktop computer. As a comparison, ECPP is not very demanding on space. In order to make the algorithm available on a desktop PC, space-efficient exponentiation of $1 + x$ is highly desirable.

Our algorithm is quite flexible. For the sake of theoretical clarity, we use just one round of ECPP reduction in the algorithm. To implement the algorithm, it may be better to follow the ECPP and launch the iteration of AKS as soon as an intermediate prime becomes good. Again assuming that the intermediate primes are distributed randomly, the expected number of rounds will be $\log\log n$. It could be a better strategy, since the intermediate primes get smaller.

We can certainly incorporate small time-saving features suggested by various researchers on the original version of AKS. It involved a more careful study of the lower bound and the upper bound of $|G_n|$. We refer to the paper by Bernstein [7] for details.

Remark. Recently Bernstein, and independently Mihailescu and Avanzi, announced a random primality-proving algorithm which has provable time complexity $\tilde{O}(\log^4 n)$. Their result dramatically improves the Adleman–Huang [1] algorithm, which was the only primality-proving algorithm known before AKS with provable random polynomial-time complexity. Their result is based on Berrizbeitia's idea as well. Similar to our algorithm, the most time-consuming part in their algorithm is to compute $(1 + x)^e \pmod{m, x^r - a}$. Suppose the input is an integer n. In our algorithm, $e \leq n$ and $r \leq 2(\log n)^2$, while in their algorithm, $e = n^d$ for some small integer d, and $r = O(d(\log n)^2)$. Hence in practice, for a general prime, our algorithm is several times faster than theirs in the verification part, assuming that the scale multiplication of a point on an elliptic curve takes negligible time. Furthermore, most parameters in our algorithm can be adjusted, and hence the algorithm is more flexible.

Acknowledgments

We thank Professor Carl Pomerance for helping to prove Theorem 2 and Professor Pedro Berrizbeitia for very helpful discussions.

References

[1] L. M. Adleman and M. A. Huang. *Primality Testing and Abelian Varieties Over Finite Fields*. Lecture Notes in Mathematics 1512. Springer-Verlag, Berlin, 1992.

[2] L. M. Adleman, C. Pomerance, and R. S. Rumely. On distinguishing prime numbers from composite numbers. *Annals of Mathematics*, 117:173–206, 1983.

[3] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. *Annals of Mathematics*, 160(2):781–793, 2004.

[4] A.O.L. Atkin. Lecture notes of a conference in Boulder (Colorado), 1986.

[5] A.O.L. Atkin and F. Morain. Elliptic curves and primality proving. *Mathematics of Computation*, 61:29–67, 1993.

[6] E. Bach and J. Shallit. *Algorithmic Number theory*, volume I. The MIT Press, Cambridge, MA, 1996.

[7] D. J. Bernstein. Proving primality in essentially quartic random time. http://cr.yp.to/, 2003.

[8] P. Berrizbeitia. Sharpening "PRIMES is in P" for a large family of numbers. *Mathematics of Computation*, 74(252):2043–2059, 2005.

[9] S. Goldwasser and J. Kilian. Almost all primes can be quickly certified. In *Proc*. 18*th ACM Symp*. *on Theory of Computing*, pages 316–329, Berkeley, CA, 1986.

[10] E. Kaltofen, T. Valente, and N. Yui. An improved Las Vegas primality test. In *Proc*. 1989 *Internat*. *Symp*. *on Symbolic Algebraic Computing*, pages 26–33. ACM Press, New York, 1989.

[11] A. Lenstra and H. W. Lenstra Jr. *Handbook of Theoretical Computer Science A*, pages 673–715. Elsevier and MIT Press, Amsterdam and Cambridge, MA, 1990.

[12] F. Morain. Primality proving using elliptic curves: An update. In *Proceedings of ANTS III*, pages 111–127. Lecture Notes in Computer Science, volume 1423. Springer-Verlag, Berlin, 1998.