A Polynomial Time Algorithm for Prime Recognition

Riaal Domingues

Submitted in partial fulfillment of the requirements for the degree
Magister Scientiae
in the Department of Mathematics and Applied mathematics
in the Faculty of Natural and Agricultural Sciences
University of Pretoria
Pretoria

17 January 2006

Abstract

Prime numbers are of the utmost importance in many applications and in particular cryptography. Firstly, number theory background is introduced in order to present the non-deterministic Solovay-Strassen primality test. Secondly, the deterministic primality test discovered by Agrawal, Kayal and Saxena in 2002 is presented with the proofs following their original paper. Lastly, a remark will be made about the practical application of the deterministic algorithm versus using the non-deterministic algorithms in applications.

Contents

Chapter 1

Introduction

Prime numbers have been studied for centuries and must be the most important numbers known to us. Prime numbers have both theoretical and practical applications with the most important practical application perhaps in cryptography. Asymmetrical key crypto systems rely on the fact that it is difficult to factor a very large given composite number into its prime factors. In order for such a system to be practical however, the designer or user of the system should start off with large prime numbers (which should be kept secret) that are multiplied to form a large composite number that can be made public without compromising the crypto system. To identify a given large number (say 100 digits or more) as a prime number is thus of great importance.

This dissertation presents the deterministic algorithm discovered by Agrawal, Kaya and Saxena in 2002 (generally referred to as the AKS algorithm). Some number theory is also developed to introduce the well known Solovay-Strassen non-deterministic primality test as a contrast to provide a feel for the practical usability of the AKS algorithm.

Some fundamental number theory results will be given here, but prior knowledge of abstract group theory is assumed, including some results from finite fields. For a background on group theory and finite fields one can reference [4]. For an introduction to number theory one can reference [5] for easy reading, and for a more thorough introduction [6] would suffice. For a background on non-deterministic algorithms [7] is a good reference. The original paper on the AKS deterministic polynimial time algorithm is that of [1] and was used for this dissertation. The official paper was published in [3]. There are also some papers and books that followed, such as the paper by Granville [9]. All the algorithms required, such as an algorithm to determine whether a given integer is a perfect power, can be found in [8].

Many publications are available on the Internet and in hard copy, presenting the deterministic algorithm and its proof. Many of these publications prove the correctness of the algorithm using alternative approaches instead of following the original proof. This dissertation intends to follow the original proof.

1.1 History

Prime numbers have been studied intensively since the days of Euclid and Eratosthenes. The existence of a deterministic primality test has not really been under question as it is easy to see that the following algorithm determines deterministically whether a given number is prime.

Algorithm 1 Input: *Natural number n to be tested for primality.* **Output:** *PRIME or COMPOSITE.*

1. *For $i = 2$ to $\lfloor \sqrt{n} \rfloor$ do*
2. *If n is divisible by i then halt and* **return** *COMPOSITE.*

3. **return** *PRIME.*

The problem with this algorithm however, is that it is impractical. Given very large integers (of say 100 digits or more), we want to test for primality in a fairly short time, without testing whether all the preceding integers divide this integer. This algorithm runs in exponential time, i.e. if the size of the input integer n doubles, the time required to determine whether it is prime grows exponentially.

To make this more precise from a complexity point of view, we can consider the running time of the algorithm using the big-O notation. The number of digits in n will be given by $\lfloor log\ n \rfloor$, where the base of log can be taken to express the number of digits of choice (i.e. 2 for binary, 10 for representation as integer, etc.). We will thus have to test $2^{\frac{\lfloor log\ n \rfloor}{2}}$ integers in Algorithm 1 in the worst case before we can determine that n is prime (the worst case being when n is prime). Clearly this amounts to a running time of $O(2^{\frac{\lfloor log\ n \rfloor}{2}})$, which is exponential. What we want to achieve, is to find an algorithm that runs in polynomial time, i.e. the running time should be $O(log^c\ n)$ where c is some constant integer.

Fortunately, results from Fermat and Euler present other ways of testing for primality. These tests are however of a non-deterministic nature which means that they rely on probability (even though they do run in polynomial time). The non-deterministic tests can output PRIME for a composite number with a (chosen) small probability. There is also quite some number theory that one needs to develop first, before it can be proven that these tests are actually successful. The most state of the art algorithm to date for primality testing is based on eliptic curve methods which involves even deeper results in number theory. The most well-known and most used non-deterministic polynomial time tests are the Solovay-Strassen and the Miller-Rabin tests.

Before the year 2002, the best deterministic algorithm to determine whether n is prime had a running time of $O(log^{log\ log\ log\ n}\ n)$ ([9], section 1.1). Clearly this is still in exponential time, even though the $log\ log\ log\ n$ term grows "fairly slowly".

1.2 The breakthrough

Agrawal, Kayal and Saxena of the Indian Institute of Technology in Kanpur, published their breakthrough on the Internet in a preliminary paper in August 2002. They found (and proved valid) a deterministic algorithm for primality testing that executes in polynomial time. This was a major breakthrough and the correctness thereof was doubted for a short while. What was so remarkable about the algorithm, was both its simplicity as well as the simplicity of the proof. This algorithm was proved to have a running time of $O(log^{10,5}\ n)$. Some improvements were made by Lenstra and Pomerance to $O(log^6\ n)$ ([9], paragraph 7.1) since the discovery. Further improvements to the algorithm were also made by Berrizbeitia and Berstein ([9], section 6), to bring the running time down to as little as $O(log^4\ n)$.

Kayal and Saxena had been students of Agrawal at the time. Agrawal received the Clay Research Award of the Clay Mathematical Institute which is presented "as its highest recognition of general achievement in mathematical research to one or more mathematicians". Agrawal and his students were invited to the presentation ceremony, however, his student's visas were denied by the U.S. State Department on the grounds that "they gave insufficient proof that after their one week visit to the United States, they would return to India" [2].

Chapter 2

Elementary number theory

The non-deterministic primality test requires number theory background in order to understand why the algorithm works so well. We will introduce the essentials in this chapter as well as prove the following theorem.

Theorem 2 (Euler's Criterion) *Let n be an odd integer. Then n is prime if and only if*

$$\left(\frac{a}{n}\right) \equiv a^{\frac{n-1}{2}} \ (mod\ n)$$

for all $a \in \mathbb{Z}_n^\times$.

Here $\left(\frac{a}{n}\right)$ denotes the Jacobi symbol. This will form the basis of the Solovay-Strassen test. We will now introduce the necessary number theory in order to prove this theorem.

2.1 Congruence

Congruence is one of the most useful tools in basic number theory. It was introduced by Karl Friedrich Gauss in the nineteenth century.

2.1.1 Some basics

Many of the basic theorems and definitions regarding congruence will be familiar to the reader. It is given here (some without proof) for the sake of completeness.

Theorem 3 (Division Algorithm) *Let a and b be integers with $b > 0$. Then there exist unique integers q and r such that $a = bq+r$ where $0 \leq r < b$.*

Definition 4 *Let n be a positive integer. For any two integers a and b we say that a* **is congruent to** *b modulo n if $n \mid (a-b)$ and we write $a \equiv b \ (mod\ n)$.*

Definition 5 *Given any two integers a and b with $b > 0$ we can use the division algorithm to write $a = bq + r$ with $0 \leq r < b$ with $q \in \mathbb{N}$. We call r* **the least non-negative residue of** *a modulo n.*

This gives us the basis to form residue classes. By the above definition it is clear that every integer is congruent to exactly one $r \in \{0, 1, 2, \ldots, n-1\}$. Each of the values $0, 1, 2, \ldots, n-1$ forms a residue class, and any integer that is congruent to one of these integers $i \in \{0, 1, 2...n-1\}$ is said to belong to the residue class $i \ (mod\ n)$. It is customary to denote the residue classes by the representative, which is the least non-negative residue, and the notation $[i]$ is used to represent the residue class i. Residue classes are important because many properties of numbers can be proved by proving it for the residue class *modulo* n it belongs to.

Definition 6 *A* **complete system of residues** *modulo n is a set of integers such that every integer is congruent modulo n to exactly one integer of the set.*

Therefore $\{0, 1, 2, \ldots, n-1\}$ is a complete system of residues modulo n.

Some properties of congruences follow.

Theorem 7 *Let a, b, c and n be integers with $n > 0$ with $a \equiv b \ (mod\ n)$. Then*

$$a + c \equiv b + c \ (mod\ n) \tag{2.1}$$

$$ac \equiv bc \ (mod\ n). \tag{2.2}$$

Cancelling common factors in congruences are not always allowed. Consider for example the congruence $3 \times 4 \equiv 5 \times 4 \ (mod\ 8)$. Clearly, if we cancel the common factor 4, the new "congruence" is not a congruence anymore, i.e. $3 \not\equiv 5 (mod\ 8)$. We now present a theorem to show when it is allowed to cancel common factors in congruences.

Theorem 8 *Let a, b, c and n be integers with $n > 0$, $ac \equiv bc \ (mod\ n)$ and $d = gcd(c, n)$. Then $a \equiv b \ (mod\ \frac{n}{d})$.*

Proof. Since $ac \equiv bc \ (mod\ n)$ there exists an integer k such that

$$ac - bc = kn$$

and therefore that

$$(a - b)(c/d) = k(n/d).$$

Since $d = gcd(n, c)$, n/d has no common factors with c/d which means that n/d divides $a - b$ and the result follows. ■

2.1.2 Solving congruences

Next we look at solving congruences. Throughout the text x will denote the unknown to be solved.

One of the most basic congruences with an unknown is the following:

$$ax \equiv b \ (mod \ n).$$

We regard all the solutions from the same congruence class as the same solution.

Example 9 *The numbers* $2, 7, 12, 17, \ldots$ *are all the solutions for the congruence* $3x \equiv 1 \ (mod \ 5)$. *However, all these numbers belong to the congruence class* 2 *modulo* 5 *and hence there is only one solution to this congruence.*

The following example illustrates that a congruence does not necessarily need to have a solution.

Example 10 *The congruence* $5x \equiv 6 \ (mod \ 15)$ *has no solutions. To see this, one only needs to realise that* $5x$ *is a multiple of* 5, *and hence the only residue classes* $5x$ *can belong to are* 0, 5 *and* 10.

However, there is a theorem to assist in identifying when a congruence has a solution, and given one solution will assist in finding all the solutions.

Definition 11 *An integer* a *is called a* **unit** *modulo* n *if the congruence* $ax \equiv 1 (mod \ n)$ *has a solution.*

The following theorem identifies which integers *modulo* n are units.

Theorem 12 *Let a and n be integers. Then a is a unit modulo n if and only if $gcd(a, n) = 1$.*

From algebra we know that the units *modulo* n form a group. This leads to the definition of the unit group $\mathbb{Z}_n^\times$.

Definition 13 *The unit group of $\mathbb{Z}_n$ is given by the residue classes modulo n which are units, hence*

$$\mathbb{Z}_n^\times = \{a \in \mathbb{Z}_n \mid gcd(a, n) = 1 \text{ and } 1 \leq a \leq n\}.$$

The unit group is therefore all the elements that are relatively prime to n and it follows that the group's order is given by the Euler totient function, i.e. $|\mathbb{Z}_n^\times| = \phi(n)$. Now we have introduced enough concepts to prove the following important theorem regarding the solutions of the linear congruence $ax \equiv b \ (mod \ n)$.

Theorem 14 *Let a, b and n be integers with $n > 0$. Also, set $d = gcd(a, n)$. Then the congruence $ax \equiv b \ (mod \ n)$ has solutions if and only if $d \mid b$. In this case, given that x_0 is a solution, there will be exactly d solutions and these solutions are given by $x_0 + n'$, $x_0 + 2n'$,...,$x_0 + (d-1)n'$ where $n' = n/d$.*

Proof. Let us first prove that $ax \equiv b \ (mod \ n)$ has solutions if and only if $d \mid b$.

Firstly, if $ax \equiv b \ (mod \ n)$ with $d = gcd(a, n)$, clearly $d \mid b$.

Next suppose $d \mid b$. Since d divides all the factors in the congruence, it follows that

$$(\frac{a}{d})x \equiv \frac{b}{d} \ (mod \ \frac{n}{d}) \quad \text{i.e.} \tag{2.3}$$

$$a'x \equiv b' \ (mod \ n') \tag{2.4}$$

where $a' = \frac{a}{d}$, $b' = \frac{b}{d}$ and $n' = \frac{n}{d}$. Now $gcd(a', n') = 1$ and a' is a unit *modulo* n'. If we multiply by the inverse of a', it is clear that we will solve for x *modulo* n' in (2.4). But it is clear that a solution of (2.4) is also a solution of the original congruence and we have proved that $ax \equiv b \ (mod \ n)$ has a solution. It means that $ax \equiv b \ (mod \ n)$ is equivalent to $a'x \equiv b' \ (mod \ n')$. Furthermore, all the solutions of $a'x \equiv b' \ (mod \ n')$ belong to the same congruence class *modulo* n'.

It remains to prove that the number of solutions are d. Let x_0 be a solution of (2.4) and consider

$$x_k = x_0 + kn' \tag{2.5}$$

where k is an integer. For any integer k we know that x_k is in the same residue class as x_0 *modulo* n', and hence is a solution of $ax \equiv b \ (mod \ n')$. However, for $ax \equiv b \ (mod \ n)$ the solutions given by (2.5) are not all the same. For $k = 0, 1, 2, ..., d-1$, every x_k formed by (2.5) is in a different equivalence class *modulo* n. In fact $x_0, x_1, \ldots, x_{d-1}$ are all the distinct solutions of $ax \equiv b \ (mod \ n)$.

■

The following example illustrates the use of this theorem.

Example 15 *Consider* $5x \equiv 10 \ (mod \ 15)$. *We notice that* $d = gcd(5, 15) = 5$ *divides* 15 *and thus the congruence* $5x \equiv 10 \ (mod \ 15)$ *is in fact solvable. To solve the congruence we first divide by* 5 *to get*

$$x \equiv 2 \ (mod\ 3).$$

We are thus looking for an x such that $x - 2 = 3k$ where k is some integer. Setting $x = 5$ will solve this equation and all the solutions will be given by $x_k = 5 + 3k$, where $k \in \{0, 1, 2, 3, 4\}$. See Table 2.1.

k	0	1	2	3	4
Solution	5	8	11	14	17
Solution (mod 15)	5	8	11	14	2

Table 2.1: Solutions of the congruence $5x \equiv 10 \ (mod\ 15)$

Corollary 16 *Let a, b and n be integers with $n > 0$. Also, set $d = gcd(a, n)$. There are exactly $\frac{n}{d}$ residue classes x modulo n satisfying $ax \equiv b \ (mod\ n)$.*

We also want to consider the solvability in abstract terms. Let G be a multiplicative cyclic group of order n. Let $z, b \in G$ be any elements and a any natural number. We will consider solutions of the equation $z^a = b$ where z is the unknown element.

Theorem 17 *Let G be a cyclic multiplicative group of order n. For a given $b \in G$ and a natural number a, the equation $z^a = b$ is solvable in G, if and only if $b^{\frac{n}{d}} = 1$ where $d = gcd(a, n)$, and in that case there are exactly d solutions.*

Proof. Suppose $z^a = b$. Then

$$\begin{aligned} b^{\frac{n}{d}} &= (z^a)^{\frac{n}{d}} \\ &= (z^{\frac{a}{d}})^n \\ &= 1. \end{aligned}$$

For the converse suppose $b^{\frac{n}{d}} = 1$. Let α be a generator of G. Then $b = \alpha^l$ for some l, say. We want to show that there exists an integer k such that $(\alpha^k)^a = \alpha^l$. Such a k will exist if the congruence

$$ka \equiv l \ (mod \ n)$$

is solvable for k. But by Theorem 14 we know this congruence is solvable if and only if $d \mid l$. But since $(\alpha^l)^{\frac{n}{d}} = 1$, we have that $n \mid l\frac{n}{d}$, i.e. $d \mid l$ and we are done.

■

The Chinese Remainder Theorem is a very handy theorem that is often used in the proofs.

Theorem 18 (Chinese Remainder Theorem) *Consider $n_1, \cdots, n_k$ where each $n_i \in \mathbb{N}$ and $gcd(n_i, n_j) = 1$ for $1 \leq i < j \leq k$. Then for any given integers $c_1, \cdots, c_k$ the congruences*

$$\begin{aligned} x &\equiv c_1 \ (mod \ n_1) \\ x &\equiv c_2 \ (mod \ n_2) \\ &\vdots \\ x &\equiv c_k \ (mod \ n_k) \end{aligned}$$

have a unique solution modulo n where $n = n_1 n_2 \cdots n_k$.

Proof. Let $m_j = \frac{n}{n_j}$ for $1 \leq j \leq k$. Then $gcd(m_j, n_j) = 1$ for $j \neq i$ and hence we can find a_j and b_j such that $a_j m_j + b_i n_j = 1$. From this follows that

$a_j m_j \equiv 1 \ (mod\ n_j)$. Thus we can choose x_j such that $m_j x_j \equiv c_j \ (mod\ n_j)$. Now let $x = m_1 x_1 + \cdots + m_k x_k$. Then x has the desired properties. It is easy to see that the solution is unique. Suppose $x \equiv y \ (mod\ n_j)$ for $1 \leq j \leq k$, then surely $x \equiv y \ (mod\ n)$ where $n = n_1 n_2 \cdots n_k$. ■

2.2 A special congruence: The quadratic residue

The Jacobi and Legendre symbols are both related to the solvability of the congruence

$$x^2 \equiv a \ (mod\ n).$$

Definition 19 (Quadratic residue) *Let a and n be natural numbers and $gcd(a, n) = 1$. Then a is called a* **quadratic residue** *modulo n if the congruence $x^2 \equiv a (mod\ n)$ is solvable and a is called a* **quadratic non-residue** *if this congruence has no solution.*

Example 20 *Let us consider the congruence $x^2 \equiv 5 (mod\ 7)$.*

x	0	1	2	3	4	5	6
$x^2 (mod\ 7)$	0	1	4	2	2	4	1

Table 2.2: The squares $(mod\ 7)$

Table 2.2 lists all the squares $(mod\ 7)$. It is clear that none of them is 5 and hence 5 is a quadratic non-residue modulo 7. However, the congruence $x^2 \equiv 2 (mod\ 7)$ is solvable. In fact it has two solutions namely $x = 3$ and $x = 4$ as can be seen from Table 2.2.

Definition 21 (Legendre symbol) *Let p be a prime and a any integer such that $p \nmid a$. Then*

$$\left(\frac{a}{p}\right) = \begin{cases} 1, & \text{if } a \text{ is a quadratic residue modulo } p \\ -1, & \text{if } a \text{ is a quadratic non-residue modulo } p. \end{cases}$$

Example 22 *By using Table 2.2 we can see that the Legendre symbols have the values*

$$\left(\frac{1}{7}\right) = \left(\frac{2}{7}\right) = \left(\frac{4}{7}\right) = 1$$
$$\left(\frac{3}{7}\right) = \left(\frac{5}{7}\right) = \left(\frac{6}{7}\right) = -1.$$

Theorem 23 ((Part of) Euler's criterion) *Let p be an odd prime and a any natural number. Then*

$$\left(\frac{a}{p}\right) \equiv a^{\frac{1}{2}(p-1)} \ (mod\ p).$$

Proof. $\mathbb{Z}_p^\times$ is cyclic and $|\mathbb{Z}_p^\times| = p - 1$. From Theorem 17 we know that $z^2 = a$ is solvable if and only of $a^{\frac{p-1}{d}} = 1$, where $d = gcd(2, p-1) = 2$. We thus have that

$$\left(\frac{a}{p}\right) = 1 \text{ if and only if } a^{\frac{p-1}{2}} \equiv 1 \ (mod\ p).$$

Take any a that is relatively prime to p. Then

$$a^{p-1} - 1 = (a^{\frac{p-1}{2}} - 1)(a^{\frac{p-1}{2}} + 1) \equiv 0 \ (mod\ p).$$

From this we can see that $a^{\frac{p-1}{2}}$ can only attain two values namely

$$a^{\frac{p-1}{2}} \equiv 1 \ (mod\ p) \ or \ a^{\frac{p-1}{2}} \equiv -1 \ (mod\ p).$$

Thus

$$\left(\frac{a}{p}\right) = -1 \text{ if and only if } a^{\frac{p-1}{2}} \equiv -1 \ (mod\ p).$$

■

Corollary 24 *The Legendre symbol is multiplicative, i.e. for integers a and b not divisible by an odd prime p*

$$\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right).$$

2.3 The Jacobi symbol

The Jacobi symbol is a generalisation of the Legendre symbol in the sense that we allow a composite number instead of a prime number.

Definition 25 (Jacobi symbol) *Let n be a positive odd integer such that $n = p_1 \cdots p_k$, p_i prime and not all necessarily distinct. Let a be any integer such that $gcd(a, n) = 1$. The Jacobi symbol is then defined as*

$$\left(\frac{a}{n}\right) = \begin{cases} 1 & \text{if } a = 1 \\ 0 & \text{is } a = 0 \\ \left(\frac{a}{p_1}\right) \cdots \left(\frac{a}{p_k}\right) & \text{otherwise.} \end{cases}$$

Careful consideration is now neccessary. When $\left(\frac{a}{n}\right) = 1$, it does not necessarily mean that a is a quadratic residue. In fact we have the following result.

Theorem 26 *An integer a is a quadratic residue modulo n if and only if a is a quadratic residue modulo p for every prime p in the prime factorisation of n.*

Proof. Let p be any prime divisor of n. Suppose $\left(\frac{a}{n}\right) = 1$. By definition there exists an x such that $x^2 \equiv a \ (mod\ n)$. But then $x^2 \equiv a \ (mod\ p)$ for all prime divisors p of n.

For the converse suppose that $\left(\frac{a}{p}\right) = 1$ for all prime divisors p of n. Set $f(x) = x^2 - a$. We want to show by induction that $f(x) \equiv 0 \ (mod\ p^j)$ has a solution for all prime divisors p of n and for all $j \geq 1$. The case $j = 1$ is trivial since $\left(\frac{a}{p}\right) = 1$ by assumption and hence $f(x) \equiv 0 \ (mod\ p)$. Suppose y is such that $f(y) \equiv 0 \ (mod\ p^j)$. Then for any integer z

$$\begin{aligned} f(y + p^j z) &= y^2 + 2yp^j z + p^{2j} z^2 - a \\ &\equiv f(y) + p^j 2yz \ (mod\ p^{j+1}). \end{aligned}$$

By Theorem 14 we know that the congruence $2yz + \frac{f(y)}{p^j} \equiv 0 \ (mod\ p)$ is solvable for z if $gcd(2y, p) \mid \frac{f(y)}{p^j}$. Since p is odd, $gcd(2y, p) = 1$ or $gcd(2y, p) = p$. However $y^2 - a \equiv 0 \ (mod\ p^j)$ which implies $y^2 - a \equiv 0 \ (mod\ p)$ so that $gcd(2y, p) = 1$. It now follows that we can find a z such that $2yz + \frac{f(y)}{p^j} \equiv 0 \ (mod\ p)$. Then $f(y + p^j z) \equiv 0 \ (mod\ p^{j+1})$ and hence we have shown that there exists a solution to the congruence $x^2 - a \equiv 0 \ (mod\ p^j)$ for all prime divisors p of n and for all $j \geq 1$. We now have a system of congruences, for

the given a:

$$x^2 \equiv a \ (mod \ p_1^{j_1})$$
$$\vdots$$
$$x^2 \equiv a \ (mod \ p_k^{j_k})$$

where $n = p_1^{j_1} p_2^{j_2} \dots p_k^{j_k}$ is the prime factorisation of n. By the Chinese Remainder Thereom there exists a unique solution *modulo* n and we are done.

■

The Jacobi symbol, like the Legendre symbol, is multiplicative, i.e.

$$\left(\frac{ab}{n}\right) = \left(\frac{a}{n}\right)\left(\frac{b}{n}\right)$$

for all integers a and b. Furthermore for any integers m and n such that $gcd(m, n) = 1$ we also have

$$\left(\frac{a}{mn}\right) = \left(\frac{a}{m}\right)\left(\frac{a}{n}\right).$$

These properties all follow easily from the fact that the Legendre symbol is multiplicative.

We are now in a position to prove Euler's Criterion:

Theorem 27 (Euler's Criterion) *Let n be an odd integer. Then n is prime if and only if*

$$\left(\frac{a}{n}\right) \equiv a^{\frac{n-1}{2}} \ (mod \ n)$$

for all $a \in \mathbb{Z}_n^\times$.

Before we prove Euler's criterion, we first prove the following lemma by Monier used in die proof of Euler's criterion.

Lemma 28 *Let n be an odd integer and let $p_1, \cdots, p_k$ be the distinct prime factors of n. Then*

$$|\{a \in \mathbb{Z}_n^\times : a^{\frac{n-1}{2}} \equiv (\frac{a}{n}) \ (mod\ n)\}| = \delta \prod_{i=1}^{r} gcd(\frac{n-1}{2}, p_i - 1)$$

where $\delta \in \{\frac{1}{2}, 1, 2\}$.

Proof. Define the homomorphisms $f, g, h : \mathbb{Z}_n^\times \to \mathbb{Z}_n^\times$ by:

$$f(b) = b^{\frac{n-1}{2}}$$
$$g(b) = \left(\frac{b}{n}\right)$$
$$h(b) = b^{\frac{n-1}{2}} \left(\frac{b}{n}\right)$$

where $b \in \mathbb{Z}_n^\times$. We want to determine the cardinality of the kernel of h. Note that

$$b \in ker\ h \text{ if and only if } f(b) = g(b) = 1 \ or\ f(b) = g(b) = -1.$$

Set $M_1 = \{b \in \mathbb{Z}_n^\times : f(b) = g(b) = 1\}$ and
$M_2 = \{b \in \mathbb{Z}_n^\times : f(b) = g(b) = -1\}$. Then

$$ker\ h = M_1 \cup M_2 \ and\ M_1 \cap M_2 = \emptyset.$$

Now two possiblities exist. Either $M_2 = \emptyset$ or $M_2 \neq \emptyset$.

$M_2 \neq \phi$: For any $z_0 \in M_2$ define the map $z \mapsto z_0 z$ from M_1 to M_2. This is a bijective map: It is trivial to see that the map is injective. To prove that the

map is surjective, we prove that for any $x \in M_2$, $x^{-1} \in M_2$. $xx^{-1} = 1 \in M_1$. So consider

$$\begin{aligned} (xx^{-1})^{\frac{n-1}{2}} &= (1)^{\frac{n-1}{2}} \\ \iff x^{\frac{n-1}{2}}(x^{-1})^{\frac{n-1}{2}} &= 1 \\ \iff -1(x^{-1})^{\frac{n-1}{2}} &= 1 \\ \iff (x^{-1})^{\frac{n-1}{2}} &= -1. \end{aligned}$$

We also need to show that $\left(\frac{x^{-1}}{n}\right) = -1$. Firstly note that $\left(\frac{xx^{-1}}{n}\right) = \left(\frac{1}{n}\right) = 1$. Recalling that the Jacobi symbol is multiplicative, we thus have

$$\begin{aligned} 1 &= \left(\frac{xx^{-1}}{n}\right) \\ &= \left(\frac{x}{n}\right)\left(\frac{x^{-1}}{n}\right) \\ &= -1\left(\frac{x^{-1}}{n}\right). \end{aligned}$$

Therefore, we have proved that inverses of elements of M_2 are also in M_2. The same method of proof will show that $z_0^{-1}x \in M_1$ and thus the map is surjective. Since the map is bijective and $M_2 \neq \phi$, $|M_1| = |M_2|$ and it follows that

$$|ker\ h| = \epsilon|M_1|$$

where $\epsilon \in \{1, 2\}$.

We also have the following:

$[f(b) = 1]$ if and only if $[f(b) = 1\ and\ g(b) = 1]$ *or* $[f(b) = 1\ and\ g(b) = -1]$

which results in a corresponding partition:

$$ker\ f = M_1 \cup M_1'$$

where $M_1' = \{b \in \mathbb{Z}_n^\times : f(b) = 1\ and\ = g(b) = -1\}$.

Once again either $M_1' = \phi$ or $|M_1| = |M_1'|$, so $|ker\ f| = \gamma|M_1|$ for some $\gamma \in \{1, 2\}$. Thus,

$$|ker\ h| = \epsilon\gamma^{-1}|ker\ f|.$$

Therefore, if we can show that

$$|ker\ f| = \prod_{i=1}^{r} gcd(\frac{n-1}{2}, p_i - 1)$$

the result will follow and we are done.

For $i = 1, \ldots, r$ set $q_i = p_i^{\alpha_i}$, where α_i is the largest power of p_i in the prime factorisation of n. Set $b_i^{\frac{n-1}{2}} \equiv 1\ (mod\ q_i)$. Then we have the following system of congruences:

$$\begin{aligned} b_1^{\frac{n-1}{2}} &\equiv 1\ (mod\ q_1) \\ b_2^{\frac{n-1}{2}} &\equiv 1\ (mod\ q_2) \\ &\vdots \\ b_r^{\frac{n-1}{2}} &\equiv 1\ (mod\ q_r). \end{aligned}$$

By using the Chinese Remainder Theorem, we can find a unique solution *modulo* n and hence there is a bijection between $ker\ f$ and the r-tuples

$(b_1, \cdots, b_r)$. Since n is odd, each $\mathbb{Z}_{q_i}^{\times}$ is cyclic and it follows from Theorem 17 that each b_i can only assume the values $gcd(\frac{n-1}{2}, \phi(q_i)) = gcd(\frac{n-1}{2}, p_i - 1)$.

■

Now we prove Euler's criterion.

Proof. We have already shown in Theorem 23 that when n is prime $\left(\frac{a}{n}\right) \equiv a^{\frac{1}{2}(n-1)} \ (mod\ n)$. For the converse we show that if n is odd, but not prime, then there exists some $a \in \mathbb{Z}_n^{\times}$ such that $\left(\frac{a}{n}\right) \not\equiv a^{\frac{n-1}{2}} (mod\ n)$. There are two distinct cases:

Case 1: n is squarefree. Let $n = p_1 \cdots p_r$ be the prime factorisation of n where $p_1 < p_2 < \cdots < p_r$ with $r \geq 2$. Assume $\left(\frac{a}{n}\right) \equiv a^{\frac{n-1}{2}} (mod\ n)$ for all $a \in \mathbb{Z}_n^{\times}$. Let g be a generator of the cyclic group $\mathbb{Z}_{p_1}^{\times}$. Using the Chinese Remainder Theorem, find $a \in \mathbb{Z}_n^{\times}$ such that $a \equiv g \ (mod\ p_1)$ and $a \equiv 1 \ (mod\ \frac{n}{p_1})$. Then

$$\begin{aligned}\left(\frac{a}{n}\right) &= \left(\frac{a}{p_1}\right) \cdots \left(\frac{a}{p_r}\right) \\ &= \left(\frac{g}{p_1}\right) \left(\frac{1}{p_1}\right) \cdots \left(\frac{1}{p_r}\right) \\ &= \left(\frac{g}{p_1}\right) \\ &= -1\end{aligned}$$

since g is a quadratic non-residue as follows from Theorem 23. But, by assumption $\left(\frac{a}{n}\right) \equiv a^{\frac{n-1}{2}} (mod\ n)$, thus

$$a^{\frac{n-1}{2}} \equiv -1 \ (mod\ n) \equiv -1 \ (mod\ \frac{n}{p_1})$$

which is contradictory to the choice of a.

Case 2: n contains a prime power factor, i.e. if $n = p_1^{\alpha_1} \dots p_r^{\alpha_r}$ is the prime factorisation of n then $\alpha^i > 1$ for some $1 \leq i \leq r$. Using the Lemma 28, we will consider the ratio of $|\{a \in \mathbb{Z}_n^\times : a^{\frac{n-1}{2}} \equiv \left(\frac{a}{n}\right) (mod\ n)\}|$ and $|\mathbb{Z}_n^\times| = \phi(n)$. If this ratio is less than one, we have finished.

$$\frac{1}{\phi(n)}|\{a \in \mathbb{Z}_n^\times : a^{\frac{n-1}{2}} \equiv \left(\frac{a}{n}\right) (mod\ n)\}| \tag{2.6}$$

$$= \delta \prod_{i=1}^{r} \frac{gcd(\frac{n-1}{2}, p_i - 1)}{p_i^{\alpha_i - 1}(p_i - 1)} \tag{2.7}$$

Firstly, note that since n is odd its smallest possible prime factor is 3. We thus have the following inequality for any $i > 1$:

$$p_i - 1 < p_2 \cdots p_r - 1 < \frac{p_1 \cdots p_r - 1}{2}$$

From this follows that the largest greatest common divisor possible in $gcd(\frac{n-1}{2}, p_i - 1)$ is $p_i - 1$ and thus

$$\delta \prod_{i=1}^{r} \frac{gcd(\frac{n-1}{2}, p_i - 1)}{p_i^{\alpha_i - 1}(p_i - 1)} \leq \delta \prod_{i=1}^{r} \frac{p_i - 1}{p_i^{\alpha_i - 1}(p_i - 1)}.$$

Now, for any i the right hand side of (2.7) we find that, if $\alpha_i = 1$, then $\frac{p_i - 1}{p_i^{\alpha_i - 1}(p_i - 1)} = \frac{p_i - 1}{(p_i - 1)} = 1$. Also for $\alpha_i > 1$ we find that $\frac{p_i - 1}{p_i^{\alpha_i - 1}(p_i - 1)} \leq \frac{p_i - 1}{3\ (p_i - 1)} = \frac{1}{3}$ (recall the smallest possible p_i is 3). Combining this we now know that

$$\delta \prod_{i=1}^{r} \frac{gcd(\frac{n-1}{2}, p_i - 1)}{p_i^{\alpha_i - 1}(p_i - 1)} \leq \frac{1}{3}.$$

We also know that $\delta \leq 2$, thus we find that

$$\frac{1}{\phi(n)}|\{a \in \mathbb{Z}_n^\times : a^{\frac{n-1}{2}} \equiv \left(\frac{a}{n}\right) (mod\ n)\}| \leq \frac{2}{3}$$

and we are done.

■

Chapter 3

A non-deterministic algorithm

In non-deterministic primality tests, we want to trade the absolute accuracy of the outcome for time. The idea is the following: Suppose we want to test an integer n for primality. Further suppose that $\{a_1, a_2, \ldots a_k\}$ are integers such that $2 < a_i < n$. Suppose a test T takes as input n and a_i and as output declares n PRIME or COMPOSITE. However, with non-deterministic primality tests, the output of the test is not always correct. For some given composite number n and the correct choice of a_i, $T(n, a_i)$ will output PRIME. Let us call such an a_i a liar. For the same composite integer n and another choice a_j, the output of $T(n, a_j)$ will be COMPOSITE. Let us call such an a_j a witness.

Now suppose a test T is such that, given a prime n, the probability that, given any integer a_i, $T(n, a_i)$ outputs PRIME is 1. However, given a composite integer n, the probability that, given any integer a_i, $T(n, a_i)$ outputs PRIME is $p < 1$, where p is known from theoretical analysis. To make the test practical we randomly select a_i and test it using test T. If the output is COMPOSITE, we declare n COMPOSITE. However, if the output is PRIME,

we select another a_i and test it as well. Since the probability that any random choice of a_i is a liar is p, testing k random $a_i s$ the probability that they are all liars becomes p^k. Since $p < 1$, we can make the probability for an incorrect answer as small as we please but always greater than zero.

We can check all the integers $a_i \in \{2, \ldots n-1\}$. This however would not be practical. We thus choose a certainty we want to achieve of the correctness of the output and determine how many randomly chosen $a_i s$ we need to check to achieve this. For many practical applications this works well, especially in cryptographic applications.

3.1 Practical computations

The preceding chapter introduced the number theory to provide us with a test for primality. In order for the test to be used in practice, we need to know how to easily compute the Jacobi symbol without factorising the integer n we are testing for primality (otherwise it would defy the objective). We also need to optimise other computations like exponentiation.

3.1.1 Getting arbitrary accuracy using Euler's criterion

It is not practical to test all the elements in $\mathbb{Z}_n^\times$ to obtain a result when we use Euler's criterion for primality testing. However, the following corollary to Monier's lemma provides us with the tool to trade off accuracy for time (the number of elements in $\mathbb{Z}_n^\times$ to be tested).

Corollary 29 *Let n be odd and not prime. Then*

$$|\{b \in \mathbb{Z}_n^\times : b^{\frac{n-1}{2}} \equiv \left(\frac{b}{n}\right) \ (mod\ n)\}| \leq \frac{1}{2}\phi(n).$$

Proof. Let h be defined as in Lemma 28 and let $H = ker\ h$. H is a proper subgroup of $\mathbb{Z}_n^\times$ (this follows from Theorem 27), so the index of H in $\mathbb{Z}_n^\times$ is at least two, i.e. $(\mathbb{Z}_n^\times : H) \geq 2$. By Langrange's Theorem

$$\begin{aligned}(\mathbb{Z}_n^\times : H) &= \frac{|\mathbb{Z}_n^\times|}{|H|} \\ \text{Therefore } |\ H\ | &= \frac{|\mathbb{Z}_n^\times|}{|(\mathbb{Z}_n^\times : H)|} \\ &\leq \frac{1}{2}|\mathbb{Z}_n^\times| \\ &= \frac{1}{2}\phi(n).\end{aligned}$$

■

It follows from Corollary 29 that, choosing a random $b \in \mathbb{Z}_n^\times$ when n is not prime, the probability $prob(b^{\frac{n-1}{2}} \not\equiv \left(\frac{b}{n}\right) \ (mod\ n)) \geq \frac{1}{2}$.

3.1.2 Computing the Jacobi symbol efficiently

In order to compute the Jacobi symbol $\left(\frac{a}{n}\right)$ effectively without factorising n, we need the quadratic reciprocity law as well as the quadratic character of 2. These results are presented here without proof. The proofs can be found in [10].

Theorem 30 (Quadratic reciprocity law) *Let m and n be odd integers. Then*

$$\left(\frac{m}{n}\right) = \begin{cases} \left(\frac{n}{m}\right) & \text{if } m \equiv 1 \ (mod\ 4) \text{ or } n \equiv 1 \ (mod\ 4) \\ -\left(\frac{n}{m}\right) & \text{if } m \equiv 3 \ (mod\ 4) \text{ and } n \equiv 3 \ (mod\ 4). \end{cases}$$

We need to know the Jacobi symbol for the special case $m = 2$ namely

$$\left(\frac{2}{n}\right) = \begin{cases} 1 & \text{if } n \equiv 1 \ (mod\ 8) \text{ or } n \equiv 7 \ (mod\ 8) \\ -1 & \text{if } n \equiv 3 \ (mod\ 8) \text{ or } n \equiv 5 \ (mod\ 8). \end{cases}$$

The following rules and algorithm were taken from [8], section 6.3, algorithm 6.3.3. Recalling the properties of the Jacobi symbol we can now derive a set of rules from the above to compute the Jacobi symbol without factorising n. Suppose we want to compute $\left(\frac{a}{n}\right)$, the following rules can be applied:

1. If $a > n$ then $\left(\frac{a}{n}\right) = \left(\frac{a \ (mod\ n)}{n}\right)$.
2. if $a = 0$ then $\left(\frac{a}{n}\right) = 0$.
3. if $a = 1$ then $\left(\frac{a}{n}\right) = 1$.
4. if $4 \mid a$ then $\left(\frac{a}{n}\right) = \left(\frac{4\frac{a}{4}}{n}\right) = \left(\frac{2}{n}\right)\left(\frac{2}{n}\right)\left(\frac{\frac{a}{4}}{n}\right)$.
5. if $4 \mid a$ then

$$\left(\frac{a}{n}\right) = \begin{cases} \left(\frac{\frac{a}{2}}{n}\right) \text{ if } n \equiv 1 \ (mod\ 8) \text{ or } n \equiv 7 \ (mod\ 8) \\ \left(\frac{-\frac{a}{2}}{n}\right) \text{ if } n \equiv 3 \ (mod\ 8) \text{ or } n \equiv 5 \ (mod\ 8). \end{cases}$$

6. If $a \equiv 1 \ (mod\ 4)$ or $n \equiv 1 \ (mod\ 4)$ then $\left(\frac{a}{n}\right) = \left(\frac{n \ mod\ a}{n}\right)$.
7. If $a \equiv 3 \ (mod\ 4)$ and $n \equiv 3 \ (mod\ 4)$ then $\left(\frac{a}{n}\right) = -\left(\frac{n \ mod\ a}{n}\right)$.

The following example illustrates the use of these rules to compute the Jacobi symbol.

Example 31 *Let us compute* $\left(\frac{773}{1373}\right)$ *using the rules above. The reference on the right hand side is the rule that is applied in each case.*

$$\begin{aligned}
\left(\frac{773}{1373}\right) &= \left(\frac{600}{773}\right) && \textit{by rule (6)}\\
&= \left(\frac{150}{173}\right) && \textit{by rule (4)}\\
&= -\left(\frac{75}{173}\right) && \textit{by rule (5)}\\
&= -\left(\frac{23}{75}\right) && \textit{by rule (6)}\\
&= \left(\frac{6}{23}\right) && \textit{by rule (7)}\\
&= \left(\frac{3}{23}\right) && \textit{by rule (5)}\\
&= -\left(\frac{2}{3}\right) && \textit{by rule (6)}\\
&= \left(\frac{1}{3}\right) && \textit{by rule (5)}\\
&= 1 && \textit{by rule (3)}
\end{aligned}$$

The following algorithm applies these rules in an efficient manner to compute the Jacobi symbol. The algorithm runs in linear time (see [8], Algorithm 6.3.3).

Algorithm 32 (Computing the Jacobi Symbol) Input: *Integer a, odd integer $n \geq 3$.* **Output:** $\left(\frac{a}{n}\right)$.

1. $b \leftarrow a \ (mod\ n)$.
2. $c \leftarrow n$.

3. $s \leftarrow 1$.

4. *while* $b \geq 2$ *repeat*

5. $\quad$ *while* $4 \mid b$ *repeat* $b \leftarrow b/4$

6. $\quad$ *if* $2 \mid b$ *then*

7. $\quad\quad$ *if* $c \bmod 8 \in \{3, 5\}$ *then* $s \leftarrow (-s)$.

8. $\quad\quad$ $b \leftarrow b/2$.

9. $\quad$ *if* $b = 1$ *then break.*

10. $\quad$ *if* $b \pmod 4 = c \pmod 4 = 3$ *then* $s \leftarrow (-s)$.

11. $\quad$ $b \leftarrow c \pmod b$, $c \leftarrow b$.

12. **return** $s \times b$.

3.1.3 Fast exponentiation

Another useful algorithm we will need for the primality test is fast modular exponentiation. This algorithm is known as repeated squaring. Let a, e and n be integers and suppose we want to compute $a^e \pmod n$. Further suppose that $e = e_k 2^k + e_{k-1} 2^{k-1} + \cdots + e_1 2^1 + e_0$ is the binary expansion of e.

Algorithm 33 Input: $e = e_k \ldots e_0$, *integer* n. **Output:** $a^e \pmod n$.

1. $y \leftarrow 1$.

2. $k \leftarrow \lfloor log_2 e \rfloor$.

3. *for* $i = 1$ *to* $k + 1$ *do*

4. $\quad$ $y \leftarrow y^2 a^{e_{k-i}} \pmod n$.

5. **return** y.

To see why this algorithm works, consider the following.

$$\begin{aligned} a^e &= a^{e_k 2^k + e_{k-1} 2^{k-1} + \ldots + e_1 2^1 + e_0} \\ &= a^{e_k 2^k} a^{e_{k-1} 2^{k-1}} \ldots a^{e_1 2^1} a^{e_0 2^0} \\ &= (a_k^e)^{2^k} (a_{k-1}^e)^{2^{k-1}} \ldots (a_1^e)^{2^1} (a_0^e)^{2^0} \end{aligned}$$

This algorithm has a complexity of $O(log\ n)$ multiplications ([8], Proposition 4.3.8).

3.2 The Solovay-Strassen test

We now introduce the Solovay-Strassen test based on Euler's criterion.

Algorithm 34 Input: *Odd integer* $n \geq 3$. **Output:** *PRIME or COMPOSITE.*

1. *Choose randomly* $a \in \{2, \ldots, n-2\}$.
2. *If* $a^{\frac{n-1}{2}} \left(\frac{a}{n}\right) \pmod n \neq 1$ *then* **return** *COMPOSITE.*
3. *else* **return** *PRIME.*

It follows from Corollary 29 that the probability of this test to output PRIME when n is in fact composite, is less than $\frac{1}{2}$. Thus to improve accuracy we will repeat the test m times for the same given n , each time selecting a random a to test. After m iterations the probability that the test will output PRIME for an n that is composite is in fact less than $\frac{1}{2^m}$. Thus to get a 99% accuracy, we only need to test 69 randomly chosen integers a, irrespective of the size of n. This yields a powerful primality test for practical applications. The Solovay-Strassen algorithm has a complexity of $\tilde{O}(log^2 n)$ bit operations ([8], Proposition 6.4.7).

Chapter 4

Deterministic polynomial time algorithm

4.1 Number and finite field theory background

We require some more results from number theory and algebra that will be used in the proof of the AKS algorithm.

Theorem 35 (Fermat's Little Theorem) *Let a be an integer and p a prime and also let $gcd(a, p) = 1$. Then*

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

Proof. Consider the integers $a, 2a, \ldots, (p-1)a$. All these integers belong to different residue classes *modulo* p. Thus multiplying the integers and reducing *modulo* p will be in the same residue class as multyplying the representatives $1, 2, \ldots, p-1$ of the residue classes and reducing *modulo* p. In

other words

$$a \cdot 2a \cdot \ldots \cdot (p-1)a \equiv 1 \cdot 2 \ldots \cdot (p-1) \ (mod\ p) \tag{4.1}$$

$$\iff (p-1)!a^{p-1} \equiv (p-1)! \ (mod\ p). \tag{4.2}$$

Since $gcd((p-1)!, a) = 1$, we can cancel the common factors in (4.2) and the result follows. ■

Fermat's Little Theorem is used as the basis for the Miller-Rabin non-deterministic primality test which is not discussed here.

The following result about the least common multiple is useful.

Lemma 36 *Let $m \geq 7$. Then*

$$lcm(1, 2, \ldots, m) \geq 2^m.$$

The following results from finite field theory is given here without proof for the sake of completeness. For a proof refer to [4].

Consider the polynomial $x^r - 1$ over any field K. A root of this polynomial in K is called an r^{th} root of unity in K. Let the characteristic of K be p and suppose p does not divide r. The set of all the r^{th} roots of unity in K forms a cyclic group of order r and we denote a generator of this group by ξ. Another useful result is that if ξ is a primitive root of unity, then all the primitive roots of unity are given by ξ^s for $1 \leq s \leq r$ and $gcd(s, r) = 1$.

Definition 37 (Cyclotomic polynomial) *Let K be a field of characteristic p and let ξ be a primitive r^{th} root of unity over K. Then the polynomial*

$$Q_r(x) = \prod_{gcd(s,r)=1, 1 \leq s \leq r} (x - \xi^s)$$

is called the r^{th} cyclotomic polynomial over K.

Theorem 38 *([4], Theorem 2.47) Let d be the least positive integer such that $q^d \equiv 1 \ (mod\ r)$. Then if $K = F_q$ with $gcd(q, r) = 1$, then $Q_r(x)$ factors into $\phi(r)/d$ distinct irreducible monic polynomials in F_q of degree d.*

4.2 The AKS algorithm

We now turn our attention to the deterministic algorithm discovered by Agrawal, Kayal and Saxena in 2002. This algorithm runs in polynomial time. The algorithm is based on the following characterisation of primes.

Lemma 39 *Let $a \in \mathbb{Z}$, $n \in \mathbb{N}$, $n \geq 2$ and $gcd(a, n) = 1$. Then n is prime if and only if*

$$(x + a)^n \equiv x^n + a \ (mod\ n). \tag{4.3}$$

Proof. Let n be prime. Expanding the left hand side of (4.3) we get

$$(x + a)^n = \sum_{i=0}^{n} \binom{n}{i} x^i a^{n-i}.$$

The coefficient of x^k is thus given by $\binom{n}{k} a^{n-k}$. For the binomial coefficients $\binom{n}{k}$ for $0 < k < n$, we see that $n | \binom{n}{k}$ and hence reducing *modulo* n these terms will be zero. From Fermat's Little Theorem it follows that if n is prime then $a^n \equiv a (mod\ n)$ and $(x + a)^n \equiv x^n + a \ (mod\ n)$.

To prove the converse, suppose $n = q^l n'$, where q is a prime and $q \nmid n'$. Since $gcd(a, n) = 1$ we only need to show that n does not divide $\binom{n}{q}$ to arrive at a contradiction.

Considering the coefficient of x^q we know that $gcd(a, n) = 1$ and hence we only need to consider $\binom{n}{q}$.

$$\begin{aligned}\binom{n}{q} &= \frac{n!}{q!(n-q)!} \\ &= \frac{n(n-1)(n-2)\dots(n-q+1)}{q!} \\ &= \frac{q^{l-1}n'(n-1)(n-2)\dots(n-q+1)}{(q-1)!}\end{aligned}$$

The factors $(n-1), (n-2), \dots, (n-q+1)$ do not have q as a prime factor. To see this one only needs to realise that since n has a prime factor q it is a multiple of q. Hence the preceeding multiple of q would be $n-q$. Thus $n \nmid \binom{n}{q}$ and we have found a non-zero coefficient in the expansion of $(x+a)^n$. This implies that (4.3) does not hold and we are done. ■

This is yet another way of testing for primes. However, the problem with this test is the evaluation on the left hand side of (4.3), as it has n terms. Clearly this will not be practical for large values of n. A way of reducing the number of terms to evaluate is to reduce *modulo* $x^r - 1$ for some sufficiently "small" (and well chosen) integer r. This leads to the following definition.

Definition 40 *For any polynomials $f(x)$, $g(x)$ and $m(x)$ and any natural number n we say that $f(x) \equiv g(x)(mod\ m(x), n)$ if there exist polynomials $u(x)$ and $v(x)$ such that $f(x) - g(x) = u(x)m(x) + nv(x)$.*

The idea now is to evaluate $(x+a)^n$ reducing the terms $(mod\ x^r - 1, n)$, i.e. we hope that n is prime if and only if

$$(x+a)^n \equiv x^n + a(mod\ x^r - 1, n). \tag{4.4}$$

It is clear from Lemma 39 that all primes n will satisfy (4.4). The question is just whether some composites will also satisfy (4.4), and unfortunately the answer is yes.

Choosing the correct r will however solve this problem. Agrawal, Kayal and Saxena succeeded in the latter. We now present their algorithm. Let n be an integer to be tested for primality and let $o_r(n)$ denote *the order of* n *modulo* r, i.e. $n^k \equiv 1 \ (mod\ r)$, where k is minimal, then $o_r(n) = k$.

Algorithm 41 (AKS Algorithm) Input: *Natural number* n. **Output** *Prime or Composite.*

1. *If* $n = a^b$ *for some* $a \in \mathbb{N}$ *and* $b > 1$, *then* **return** *COMPOSITE.*
2. *Find the smallest* r *such that* $o_r(n) > 4\ log^2 n$.
3. *If* $1 < gcd(a, n) < n$ *for some* $a \leq r$ *then* **return** *COMPOSITE.*
4. *If* $n \leq r$ *then* **return** *PRIME.*
5. *For* $a = 1$ *to* $\lfloor 2\sqrt{\phi(r)} log\ n \rfloor$ *do*
6. *if* $(x + a)^n \equiv x^n + a (mod\ x^r - 1, n)$ *then* **return** *COMPOSITE.*
7. **return** *PRIME.*

Clearly, if any of the conditions above cause an output of COMPOSITE, the algorithm will halt. Step 1 of the algorithm ensures that n is not a perfect power. Step 2 finds the appropriate r of which the order *modulo* n is greater than $4\ log^2 n$ and step 4 determines that if this r is greater than n, then n must be prime. Step 3 ensures none of the a's to be tested has common factors with n for, if so, n is not prime. Step 5 is where the actual congruency testing (4.4) takes place.

What we know about n and r before step 5 is that n is not a perfect power, there exists an r such that the order of n *modulo* r is greater than $4\ log^2 n$

where $r < n$ and n has no prime factors less than r. This leads to the formulation by Andrew Granville of this algorithm into a precise mathematical theorem.

Theorem 42 *For a given integer $n \geq 2$, let $r < n$ be a positive integer such that $o_r(n) > 4\ log^2\ n$. Then n is prime if and only if*

1. *n is not a perfect power, i.e. $n \neq a^b$ for some integers a and b,*
2. *n does not have any prime factor $\leq r$,*
3. *$(x+a)^n \equiv x^n + a\ (mod\ x^r - 1, n)$ for each integer $1 \leq a \leq \sqrt{\phi(r)} log\ n$.*

Let $l = \lfloor 2\sqrt{\phi(r)} log\ n \rfloor$ for the remainder of the section to improve readability.

We first need to know that a suitable r can be found.

Lemma 43 *There exists an $r \leq \lceil 16\ log^5\ n \rceil$ such that $o_r(n) > 4\ log^2\ n$.*

Proof. Suppose that no such r exists, i.e. for all $r \leq \lceil 16\ log^5\ n \rceil$ it is true that $o_r(n) \leq 4\ log^2\ n$. Let $\{r_1, \ldots, r_t\} = \{1, 2, \ldots, \lceil 16\ log^5 n \rceil\}$. Note that this implies for every $1 \leq i \leq t$ that $n^{o_{r_i}(n)} \equiv 1\ (mod\ r_i)$. It now follows that each of these numbers r_i divides the product

$$\prod_{i=1}^{\lfloor 4\ log^2\ n \rfloor} (n^i - 1).$$

Recalling the formula $(1 + 2 + \ldots + k) = \frac{k}{2}(1 + k)$ we see that

$$\prod_{i=1}^{\lfloor 4\ log^2\ n \rfloor} (n^i - 1) < \prod_{i=1}^{\lfloor 4\ log^2\ n \rfloor} n^i$$
$$= n^{\frac{\lfloor 4\ log^2\ n \rfloor}{2}(1 + \lfloor 4\ log^2\ n \rfloor)},$$

but

$$\frac{\lfloor 4\ log^2\ n\rfloor}{2}(1+\lfloor 4\ log^2\ n\rfloor) < 2\ log^2\ n + 8\ log^4\ n$$
$$< 16\ log^4\ n,$$

thus

$$n^{\frac{\lfloor 4\ log^2\ n\rfloor}{2}(1+\lfloor 4\ log^2\ n\rfloor)} < n^{16\ log^4\ n}$$
$$= 2^{16\ log^5\ n}.$$

Since each r_i divides the product $\prod_{i=1}^{\lfloor 4\ log^2\ n\rfloor}(n^i-1)$, $lcm(r_1,\ldots,r_t) < 2^{16\ log^5\ n}$. By Lemma 36 the least common multiple of the first $\lceil 16\ log^5\ n\rceil$ numbers is at least $2^{\lceil 16\ log^5\ n\rceil}$, a contradiction. Therefore there must exist a number $r \leq \lceil 16\ log^5\ n\rceil$ such that $o_r(n) > 4\ log^2\ n$. ■

In order to prove Theorem 42, we establish a few definitions and lemmas. Firstly, let us give an outline on how we will go about proving the theorem.

Outline of the proof. Assuming n is prime and proving all the conditions hold is easy. For the converse we will assume n is composite, but all the conditions hold. We will then construct a group and find a contradiction in the size of the group, proving that our assumption was wrong in the first place. Most of the work and definitions will be required for the second part of the proof.

For the remainder of this section the following symbols will be used. Let n be composite and suppose n has a prime factor p with $n = pn'$ where n' is some integer. We also assume that all the conditions given in the theorem do hold for this composite n. Let us consider the following:

By assumption we have that

$$(x+a)^n \equiv x^n + a \ (mod\ x^r - 1, n) \text{ for all } 1 \leq a \leq l.$$

Since $p|n$ it is obvious that

$$(x+a)^n \equiv x^n + a \ (mod\ x^r - 1, p) \text{ for all } 1 \leq a \leq l. \tag{4.5}$$

But p is prime, so

$$(x+a)^p \equiv x^p + a \ (mod\ x^r - 1, p) \text{ for all } 1 \leq a \leq l. \tag{4.6}$$

We can see from this that n thus behaves like a prime in this case, and we define this property in the following way.

Definition 44 *For any polynomial $f(x)$ and natural number m, we say that m is introspective for $f(x)$ if*

$$[f(x)]^m \equiv f(x^m) \ (mod\ x^r - 1, p).$$

Note that both n and p are introspective numbers for the polynomial $x + a$ for all $1 \leq a \leq l$ (see (4.5) and (4.6)). Introspective numbers are closed under multiplication:

Lemma 45 *If m and m' are introspective for $f(x)$, then so is $m.m'$.*

Proof. Since m is introspective

$$[f(x)]^{m.m'} \equiv [f(x^m)]^{m'} (mod\ x^r - 1, p). \tag{4.7}$$

Set $x = x^m$ in (4.7) and take into account that m' is also introspective. Then

$$[f(x^m)]^{m'} \equiv [f(x^{m.m'})] \ (mod\ x^{mr} - 1, p) \tag{4.8}$$

$$\equiv [f(x^{m.m'})] \ (mod\ x^r - 1, p) \tag{4.9}$$

since $(x^r - 1)|(x^{mr} - 1)$. Combining (4.7) and (4.9) we get that $f(x)^{m.m'} \equiv f(x^{m.m'})\ (mod\ x^r - 1, p)$ as desired. ■

For a fixed number m the set of polynomials that are introspective for m is closed under multiplication.

Lemma 46 *Let m be an integer that is introspective to the polynomials $f(x)$ and $g(x)$. Then m is also introspective for the polynomial $f(x).g(x)$.*

Proof.

$$\begin{aligned}[f(x).g(x)]^m &= [f(x)]^m.[g(x)]^m \\ &\equiv f(x^m).g(x^m)\ (mod\ x^r - 1, p)\end{aligned}$$

■

Now define the following two sets:

$$I = \{n^i p^j \mid i, j \geq 0\} \tag{4.10}$$

$$P = \{\prod_{a=1}^{l}(x+a)^{e_a} \mid e_a \geq 0\} \tag{4.11}$$

It should be clear from the two preceding lemmas and the fact that both n and p are introspective numbers for the polynomial $x+a$ that all the numbers in I are introspective for all the polynomials in P. Now we define two groups that we will call G and H respectively from these sets. Firstly,

$$G = \{i\ (mod\ r) \mid i \in I\}$$

We know this is a group since $gcd(n, r) = gcd(p, r) = 1$ and all elements in I have the form $(n^i p^j)$. Let $|G| = t$. Since $o_r(n) > 4\ log^2 n$, and G is generated by n and p *modulo* r, we know that

$$t > 4\ log^2 n. \tag{4.12}$$

Secondly, let $Q_r(x)$ be the r-th cyclotomic polynomial. Let $h(x)$ be an irreducible factor of $Q_r(x)$. Then define

$$H = \{m(x)\ (mod\ h(x), p) \mid m(x) \in P\}. \tag{4.13}$$

This group is thus generated by the polynomials $x+1, x+2, \ldots, x+l$ in the field $F_q = F_p[x]/(h(x))$.

The following two lemmas provide estimates on the size of H.

Lemma 47 $\mid H \mid \geq \binom{t+l-2}{t-1}$

Proof. Let $f(x)$ and $g(x)$ be two distinct polynomials in P with degree less than t. We show that these polynomials will correspond to different polynomials in H.

Suppose $f(x) = g(x)$ in F_q. Then for $m \in I$ we also have that $[f(x)]^m = [g(x)]^m$ in F_q. But $h(x) \mid (x^r - 1)$ and m is introspective for f and g, so we have that

$$f(x^m) = g(x^m) \tag{4.14}$$

in F_q. Define the polynomial $p(y) = f(y) - g(y)$. By (4.14) x^m is a root of $p(y)$ for every $m \in G$. Also $gcd(m, r) = 1$ and hence x^m is a primitive r^{th} root of unity. Hence there will be $|G| = t$ distinct roots of $p(y)$ in F_q. However, $deg(p(y)) < t$ by the choice of $f(x)$ and $g(x)$ and Theorem 38. We have a contradiction and consequently $f(x) \neq g(x)$ in H.

Next we note that $l = \lfloor 2\sqrt{\phi(r)} log\ n \rfloor < 2\sqrt{r} log\ n < r$ and by the hypothesis $p > r$. From this follows that the polynomials $x+1, x+2, \ldots, x+l$ are all distinct in F_q. It might be possible that one of the polynomials is zero, say $x+a$. This will happen when $h(x) = x+a$. In such a case $x+a$ will not be included in H. There are thus at least $l-1$ distinct polynomials of degree 1 in H. All the polynomials in H are given by

$$(x+1)^{e_1}(x+2)^{e_2} \ldots (x+l-1)^{e_{l-1}}$$

where $e_1 + e_2 + \ldots + e_{l-1} \leq t$, and there are $\binom{t-1+l-1}{t-1} = \binom{t+l-2}{t-1}$ such polynomials. ■

Lemma 48 *If n is not a power of p, then $|H| < \frac{1}{2} n^{2\sqrt{t}}$.*

Proof. Consider the following subset of I:

$$\hat{I} = \{n^i p^j \mid 0 \leq i, j \leq \lfloor \sqrt{t} \rfloor\}. \tag{4.15}$$

Since n is not a power of p, $|\hat{I}| = (\lfloor \sqrt{t} \rfloor + 1)^2 > t$. But $|G| = t$, so there exist at least two numbers in $\hat{I}$ that are equal *modulo* r, say m_1 and m_2 with $m_1 > m_2$. We thus have that $x^{m_1} \equiv x^{m_2} (mod\ x^r - 1)$.

Let $f(x) \in P$. Then

$$\begin{aligned} [f(x)]^{m_1} &\equiv f(x^{m_1})\ (mod\ x^r - 1, p) \\ &\equiv f(x^{m_2})\ (mod\ x^r - 1, p). \end{aligned}$$

But we also have

$$f(x)^{m_2} \equiv f(x^{m_2})\ (mod\ x^r - 1, p)$$

and hence $[f(x)]^{m_1} = [f(x)]^{m_2}$ in F_q. Let $\tilde{p}(y) = y^{m_1} - y^{m_2}$. Then any $g(x) \in H$ is a root of $\tilde{p}(y)$ in F_q. There are at least $|H|$ distinct roots of $\tilde{p}(y)$ in F_q by what we have just shown. The degree of $\tilde{p}(y)$ is $m_1 \leq (np)^{\lfloor\sqrt{t}\rfloor} < \frac{1}{2}n^{2\sqrt{t}}$ and thus $|H| < \frac{1}{2}n^{2\sqrt{t}}$. ■

One can now use the bounds just proven to prove that the algorithm works correctly. This will be proven by contradiction.

Lemma 49 *If the algorithm returns PRIME, then n is prime.*

Proof. Suppose the algorithm returns PRIME. Let us make the following observations first. Considering $2\sqrt{t}\ log\ n$ and using inequality (4.12), we have

$$\begin{aligned} 2\sqrt{t}\ log\ n &\leq 2\sqrt{4log^2\ n}\ log\ n \\ &= 4\ log^2\ n \end{aligned}$$

and hence

$$t > 2\sqrt{t}\ log\ n.$$

Using Lemma 47 with $|G| = t$ and $l = \lfloor 2\sqrt{\phi(r)}\ log\ n \rfloor$ one gets

$$
\begin{aligned}
|H| &\geq \binom{t+l-2}{t-1} \\
&\geq \binom{l-1+\lfloor 2\sqrt{t}\ log\ n \rfloor}{\lfloor 2\sqrt{t}\ log\ n \rfloor} && (since\ t > 2\sqrt{t}\ log\ n) \\
&\geq \binom{2\lfloor 2\sqrt{t}\ log\ n \rfloor - 1}{\lfloor 2\sqrt{t}\ log\ n \rfloor} && (since\ l = \lfloor 2\sqrt{\phi(r)}\ log\ n \rfloor \geq \lfloor 2\sqrt{t}\ log\ n \rfloor) \\
&\geq 2^{\lfloor 2\sqrt{t}\ log\ n \rfloor} && (since\ 2\sqrt{t}\ log\ n \geq 3) \\
&\geq \frac{1}{2} n^{2\sqrt{t}}.
\end{aligned}
$$

By Lemma 48 $|G| < \frac{1}{2} n^{2\sqrt{t}}$ if n is not a power of p. Therefore, $n = p^k$ for some $k > 0$. If $k > 1$ then the algorithm will return COMPOSITE in step 1. Therefore $n = p$.

■

4.3 Complexity of the algorithm

We will refer to Algorithm 41 throughout this section.

Step 1: Determining whether n is a perfect power

This step can be determined in $\tilde{O}(log^3 n)$ bit operations and an algorithm is given in Appendix A.

Step 2: Finding r

Algorithm 50 Input: *Natural number n.* **Output:** *Smallest natural number r such that $o_r(n) > 4\ log^2 n$.*

1. $r \leftarrow 1$.
2. *while* $r < n$ *do*
3. $\quad k \leftarrow 1$.
4. $\quad$ *while* $k \leq 4\ log^2 n$ *do*
5. $\quad\quad$ *if* $n^k \equiv 1\ (mod\ n)$ *then* **return** r *and halt else* $k \leftarrow k + 1$.
6. $\quad r \leftarrow r + 1$.

For each r that is tested there are at most $O(log^2 n)$ multiplications *modulo* n, each multiplication at a cost of $\tilde{O}(log\ r)$. Testing one such potential r thus takes time $\tilde{O}(log^2 n \times log\ r)$. By Lemma 43 we know that we will find a suitable $r \leq 16\ log^5\ n$ and the inner loop will need to be executed at most $O(log^5 n)$ times. Thus, this step will take at most $\tilde{O}(log^5 n \times log^2 n \times log(16\ log^5 n)) \subseteq \tilde{O}(log^7 n)$.

Step 3: Determining whether all integers $a \leq r$ are relatively prime to n. There are r numbers to compute the greatest common divisor of, each computation taking $O(log\ n)$. Using Lemma 43 we know $r \leq 16\ log^5 n$. Consequently this step will take at most $O(r\ log\ n) = O(log^5 n \times log\ n) = O(log^6 n)$.

Step 4: Determining whether $r < n$. This step takes $O(log\ n)$.

Step 5: The final test. For the final step of the algorithm, recall that the operation of computing a single $(x+a)^n\ (mod\ x^r - 1, n)$ takes $\tilde{O}(rl(l + log\ r))$, where l is the number of bits of n (see Appendix A). Thus we have

$$\tilde{O}(r\ log\ n(log\ n + log\ r)) = \tilde{O}(r\ log^2\ n + r\ log\ n\ log\ r)).$$

We repeat this operation $2\sqrt{\phi(r)}log\ n$ times in the loop, and recalling the estimate for r (Lemma 43),

$$\begin{aligned}
&2\sqrt{\phi(r)}\ log\ n\ \tilde{O}(r\ log^2\ n + log\ n\ log\ r)\\
&= \tilde{O}(2r\sqrt{\phi(r)}\ log^3\ n + 2r\sqrt{\phi(r)}log^2\ n\ log\ r)\\
&= \tilde{O}(r^{3/2}\ log^3\ n + r^{3/2}log^2\ n\ log\ r)\\
&= \tilde{O}((log^5 n)^{3/2}\ log^3\ n + (log^5 n)^{3/2}log^2\ n\ log\ log^5 n)\\
&= \tilde{O}(log^{15/2} n\ log^3\ n + log^{15/2} n\ log^2\ n\ log\ log^5 n)\\
&= \tilde{O}(log^{10.5} n + log^{9.5} n\ log\ log^5 n)\\
&\subseteq \tilde{O}(log^{10.5} n).
\end{aligned}$$

This step dominates all the other steps and hence is the complexity of the algorithm.

It is clear that $\tilde{O}(log^{10.5} n) = O((log\ n\ log\ log\ n)^{10.5})$ is polynomial time. Since the proof does not assume anything about the base of the logarithm, this complexity is attained regardless of the representation of the integer n.

4.4 Usability of the algorithm in practical applications

The Solovay-Strassen algorithm has a complexity of $\tilde{O}(log^2 n)$ bit operations ([8], Proposition 6.4.7). Repeating the algorithm 69 times will result in an accuracy of 99% that an output of PRIME is indeed correct. This will result in a complexity of $69\tilde{O}(log^2 n) = \tilde{O}(log^2 n)$.

In the previous section it was shown that for the AKS algorithm, the complexity is $\tilde{O}(log^{10.5} n)$, which is considerably more than the non-deterministic

Solovay-Strassen test. For practical applications, the use of non-deterministic polynomial time algorithms (like the Solovay-Strassen test) are thus more desirable.

To motivate this point: Consider a system like the RSA assymetrical encryption system. The security in the system is based on obtaining two large primes p and q and multiplying them to form an integer $n = pq$. For modern applications the integer n needs to be 2048 bits long in binary representation to provide adequate security. This makes it unfeasible to try and factor n into p and q which is unknown to the adversary attacking the crypto system. Generation of these large primes typically takes 5 minutes or more on modern day computers. Although this seems to be not too bad, imagine a system that has hundreds of users. Generating these security paramaters (n, p and q) will take a very long time.

Appendix A

Computational complexity

A.1 Big O notation

In computational complexity we are not interested in the exact time it will take to run an algorithm, as technology influences this greatly. One would rather concentrate on the operations in the algorithm and we are generally interested in the number of bit operations. We now introduce the big O notation which is central to the latter. A good reference for this appendix is [11].

Definition 51 (Eventually positive partial function) *A partial function $f : \mathbb{N} \to \mathbb{R}$ is a function that needs not be defined for all $n \in \mathbb{N}$, and is called eventually positive if there exists a constant $k \in \mathbb{N}$ such that $f(n)$ is defined and strictly positive for all $n \geq k$.*

Definition 52 (Big O notation) *Let $g : \mathbb{N} \to \mathbb{R}$ be an eventually positive function. Then $O(g)$ is defined to be the set of all eventually positive functions $f : \mathbb{N} \to \mathbb{R}$ for which there exists constants $k, c \in \mathbb{N}$ such that $f(n)$ and $g(n)$ are defined for all $n \geq k$ and $f(n) \leq cg(n)$ for all $n \geq k$.*

Constant factors are not important in computational complexity and is absorbed by the big O notation, since the constant factors can be handled by the constant in the definition of the O notation. Generally, for easy representation, one can also define soft-O.

Definition 53 (Soft-O) *We write $f \in \tilde{O}(y)$ if $f \in O(y\ log^{O(1)}\ y)$ and we say f is in soft-O y.*

By $O(1)$ we mean any function that is eventually positive and bounded from above. By $log^{O(1)}\ n$, we mean *log* to the power of a function that is eventually bounded by a constant. This especially include all constant powers of $log\ n$.

A.2 Basic operations and their complexity

The complexities of the following operations are referenced here without proof. The algorithms and proofs can be found in [11] and is beyond the scope of this dissertation.

A.2.1 Addition and multiplication

Ordinary addition and subtraction of n-word integers can be done in $O(n)$. Classic multiplication of two n-word integers (i.e. as we would do it as taught in school with pencil and paper), would be quadratic, i.e. $O(n^2)$, but there exist algorithms (Fast Fourier transform) which are much faster for both multiplication and division. Both multiplication and division can be done in $\tilde{O}(n)$, i.e. linear time. We will use the latter for analysis purposes.

A.2.2 Multiplying polynomials

A basic operation in the AKS algorithm is multiplication of two polynomials and reducing modulo a polynomial, say $x^r - 1$. This is easily done by a method called single-point evaluation. Suppose we have two polynomials $f(x)$ and $g(x)$ of degree at most $r-1$ and integer coefficients between $-A$ and A for some positive integer A. We can now form a natural bijection

$$\psi : \left\{ f(x) = \sum_{i=0}^{r-1} a_i x^i \in \mathbb{Z}[x] : -A < a_i \leq A \ for\ all\ i \right\} \rightarrow \mathbb{Z}/(2A)^r.$$

Let us first show how the bijection works and then prove that it is in fact a bijection. Given a polynomial $f(x) = \sum_{i=0}^{r-1} a_i x^i$, we define

$$\psi(f) = f(2A) = a_0 + a_1(2A) + a_2(2A)^2 + \ldots + a_r(2A)^{r-1}. \tag{A.1}$$

Given a number $-A < k < A$, to find the polynomial that it represents, we first compute $a_0 \equiv k \ (mod\ 2A)$. Then $a_1 \equiv \frac{(k-a_0)}{2A} \ (mod\ 2A)$, $a_2 \equiv \frac{(k-a_0-a_1)}{(2A)^2} \ (mod\ 2A)$, and in general $a_i \equiv \frac{(k-a_0-a_1 \ldots -a_{i-1})}{(2A)^i} \ (mod\ 2A)$.

To understand this, consider again the form as given by equation (A.1). When we compute *modulo* $2A$ we "cancel" all terms that contain a factor of $2A$. However, we first subtract all the terms computed already, and then divide $(2A)^i$ so that the coefficient a_1 cannot contain a power of $2A$ and hence reducing *modulo* $2A$ will not cancel a_i , but will cancel any other coefficients left in the equation as desired.

We need to show that this in fact yields a bijection, and moreover is a homomorphism so that we can use the integers to compute the product of two polynomials fast.

To prove that the mapping ψ is a bijection, we use the fact that this mapping is onto, for given any $-A < k < A$, by the above argument one can find a polynomial $f(x)$ that will map onto k. Also, there are exactly $(2A)^r$ polynomials of degree at most $r-1$ and coefficients $-A < a_i < A$. Thus we have an onto mapping from a finite set to a finite set of the same cardinality, and hence a bijection.

To see that the mapping ψ is a homomorphism with respect to multiplication, consider $f(x) = a_0 + a_1x + \ldots + a_{r-1}x^{r-1}$ and $g(x) = b_0 + b_1x + \ldots + b_{r-1}x^{r-1}$. Their mapping to $\mathbb{Z}/(2A)^r$ will be $f(2A) = a_0 + a_1(2A) + \ldots + a_{r-1}(2A)^{r-1}$ and $g(2A) = b_0 + b_1(2A) + \ldots + b_{r-1}(2A)^{r-1}$ respectively.

Now, multiplying $f(x)g(x)$ in the polynomial ring will be exactly the same if we represent the numbers in their polynomial form in the ring $\mathbb{Z}/(2A)^r$, i.e. $f(x)g(x)$ will involve exactly the same operations in the form $f(2A)g(2A) \in \mathbb{Z}/(2A)^r$. It should be clear from this that the operation of multiplication is preserved (as will the operation of addition also be preserved, even though we are not interested in addition for this purpose).

The complexity of this algorithm is as follows. Let r be the highest degree of the two polynomials and n the largest coefficient. Then this operation can be performed in $\tilde{O}(r(log\ n + log\ r))$ ([9], Section 3b.1).

A.2.3 Greatest common divisor

Another operation that is used often is determining the greatest common divisor of two integers, say n and m where $n \geq m$. This can be done by using the Euclidean algorithm. There is a fast version of the Euclidean algorithm which can execute in $O(log\ n)$ (see [11], Chapter 11).

A.2.4 Perfect power

The following algorithm (taken from [8], Algorithm 2.3.5) takes as input an integer n and determines whether it is a perfect power, i.e. it determines whether there exist integers $a < n$ and $b > 1$ such that $n = a^b$.

Algorithm 54 Input: *Natural number* n. **Output:** *"n is a Perfect Power" or "n is not a Perfect Power".*

1. *a,b,c,m integers.*
2. $b \leftarrow 2$.
3. *while* $2^b \leq n$ *do*
4. $\quad a \leftarrow 1, c \leftarrow n$.
5. $\quad$ *while* $c - a \geq 2$ *do*
6. $\quad\quad m \leftarrow a + c \ (mod\ 2)$.
7. $\quad\quad p \leftarrow min\{m^b, n + 1\}$.
8. $\quad\quad$ *if* $p = n$ *then* **return** *"Perfect Power" and halt.*
9. $\quad\quad$ *if* $p < n$ *then* $a \leftarrow m$ *else* $c \leftarrow m$.
10. $\quad b \leftarrow b + 1$.
11. **return:** *"n is not a Perfect Power".*

The operations used in this algorithm should be the most efficient ones. In that case there are $\tilde{O}(log^2\ n)$ multiplications of the first n integers. Thus this algorithm will take at most $\tilde{O}(log^2 n)$ multiplications, each of which takes $\tilde{O}(log\ n)$. The complexity of the algorithm is then $\tilde{O}(log^2 n \times log\ n) = \tilde{O}(log^3\ n)$. The complexity analysis for this algorithm can be found in ([8], Lemma 2.3.6).

Bibliography

[1] Manindra Agrawal, Neeraj Kayal, Nitin Saxena. *PRIMES is in P*, http://www.cse.iitk.ac.in/users/manindra/primality_original.pdf.

[2] http://www.flonnet.com/fl1925/stories/20021220008613400.htm

[3] Manindra Agrawal, Neeraj Kayal, Nitin Saxena. *Primes is in P*, Anals of Mathematics 160(2), 781-793, 2004.

[4] Rudolf Lidl and Harald Niedereiter. *Introduction to finite fields and their applications revised edition*, Cambridge University press, 2000.

[5] Kenneth H. Rosen. *Elementary number theory and its applications*, Addison-Wesley publishing company, 1993.

[6] Kenneth Ireland and Michael Rosen. *A Classical Introduction to Modern Number Theory, second edition*, Springer, 1990.

[7] Song Y. Yan. *Primality testing and integer factorisation in public-key cryptography*, Kluwer Academic Publishers, 2004.

[8] Martin Dietzfelbinger. *Primality testing in polynomial time*, Springer, 2004.

[9] Andrew Granville. *It is easy to determine whether a given integer is a prime*, Bulletin of the American Mathematical Society Volume 42 Number 1, 3-38 , September 2004.

[10] Prof. W.L. Fouché. *Lecture notes*, 2002.

[11] Joachim von zur Gathen and Jurgen Gerhard. *Modern Computer Algebra second edition*, Cambridge University Press, 2003.

DECLARATION

I, the undersigned, hereby declare that the dissertation submitted herewith for the degree Magister Scientiae to the University of Pretoria contains my own, independent work and has not been submitted for any degree at any other university.

Signature:

Name: Riaal Domingues
Date: 2006-01-18