Introduction to Complexity Theory*
Lecture 11: Square roots in Z_p^* and Primality Testing

Guest Lecturer: Shafi Goldwasser
Notes taken by Yehuda Hassin, Lior Noy and Ben Stein

Spring 2001

Summary: In this lecture we introduce two randomized algorithms in number theory. The first algorithm finds in expected polynomial time the square root of a number in Z_p^* for any prime number p. We then use this algorithm to construct a polynomial time algorithm with two-sided errors for primality testing.

1 Introduction

Primality testing is known as one of the first areas for the use of randomized algorithms. Moreover, from theoretic point of view this example is still very important. Many of the problems that have randomized algorithms have also a deterministic versions. Hence, the use of randomized algorithms is usually done just for simplicity reasons or as a building stone for deterministic ones. However, there is still no deterministic polynomial time algorithm for primality testing, although we know of some randomized polynomial time algorithms for this problem for twenty five years.

First we introduce some simple definitions and basic facts in group theory.

As a first step towards primality testing we introduce an algorithm $SQRT$ for finding square roots in Z_p^*. The algorithm $SQRT$ is a *Las Vegas* algorithm, i.e., an expected polynomial time algorithm with no error probability. We then convert $SQRT$ into a *Monte Carlo* algorithm, i.e., an algorithm that runs in polynomial time but has an error probability. We use randomization in this algorithm in order to find a non square number in Z_p^*. Although **half** of the numbers in Z_p^* have no square roots, no deterministic algorithm that finds such a number is known.

We then construct an algorithm that tests weather a given number n is a prime number. Let $n = \prod_{i=1}^{k} q^{\alpha_i}$. We rely on the fact that the number of square roots for a square in Z_n^* is 2^k. We then invoke algorithm $SQRT$ to find a square root. Choosing a random solution to this equation and using the above fact (that for composite, not a prime power, there are many solutions), we introduce a two-sided error randomized algorithm for primality testing.

Although the above algorithm does not give us an absolute proof for prime numbers, in the last part of the lecture we will prove that there is a short proof, and hence the language of prime numbers is in $\mathcal{NP}$.

---

* Lecture Notes for a course given by Oded Goldreich at the Weizmann Institute of Science, Israel.

2 Definitions and Simple Facts

2.1 Definitions

We start with a definition of a group. A group G is a set of elements with a binary operation that satisfies the following properties: The binary operation is associative, has an identity element e, (i.e., for every $a \in G$, $a \circ e = a = e \circ a$), and for every $a \in G$ there exists an inverse element, denoted a^{-1}, such that $a \circ a^{-1} = e$.

Example: The additive group $Z_p = \{0, 1, 2, ..., p-1\}$, when p is an integer, with the binary operation: $a + b \stackrel{\text{def}}{=} (a+b) \bmod p$. By definition, Z_p is a group because

1. $[(a+b) \bmod p + c] \bmod p = [a + (b+c) \bmod p)] \bmod p$ (association).

2. $(a+0) \bmod p = a = (0+a) \bmod p$ (the identity element is 0).

3. $[a + (p-a)] \bmod p = 0 = [(p-a)+a] \bmod p$ (the inverse element of a is $p-a$).

Definition 1 Group order: *The order of a group G, denoted by $|G|$, is the number of elements in G.*

Denote by h^m the m-wise multiplication $h \circ \cdots \circ h$.

Definition 2 Element order: *The order of an element h in a group G is the smallest integer number m such that $h^m = 1$.*

Definition 3 Cyclic group: *A Cyclic group is a group that contains an element g such that $G = \{g, g^2, \ldots, g^{m-1}, g^m\}, m = |G|$, and g^m is the identity of the group. The element g is called the* generator *of the group.*

Example: The group Z_p is a cyclic group because $g = 1$ is a generator of the group:

$$\begin{aligned}
(1) \bmod p &= 1 \\
(1+1) \bmod p &= 2 \\
(1+1+1) \bmod p &= 3 \\
&. \\
&. \\
&. \\
\underbrace{(1+1+...+1)}_{p-1} \bmod p &= p-1 \\
\underbrace{(1+1+...+1+1)}_{p} \bmod p &= 0
\end{aligned}$$

Another example of a group that we use frequently is the group Z_p^*.

Let $Z_p^* = \{1 \leq x \leq p \mid \gcd(x,p) = 1\}$ with the binary operation being multiplication mod p, i.e., $a * b \stackrel{\text{def}}{=} (a * b) \bmod p$.

Definition 4 *The order of Z_p^* (the number of elements in Z_p^*) is known as the* Euler Totient Function*: $|Z_p^*| = \phi(p)$.*

Proposition 2.1 *If p is a prime number then $|Z_p^*| = p - 1$.*

Proof: Since for a prime p, for every $1 \leq x \leq p-1$, we know that $\gcd(x,p) = 1$. It follows simply from the definition that $|Z_p^*| = p - 1$. ∎

2.2 Some facts about Z_p^*

1. From a computational point of view, for every $a \in Z_p^*$ there is a polynomial time algorithm that finds $b \in Z_p^*$ such that $(a * b) \bmod p = 1$. This can be done, for example, by looking at the output of Extended-gcd(a, p): for $a \in Z_p^*$ the output is two integers α and β such that $\alpha p + \beta a = \gcd(a, p) = 1$. Reducing this equation mod p we get that $\beta a \equiv 1 \bmod p$, and so β is the multiplicative inverse of a.

2. *Euler's theorem:* for every $a \in Z_p^* : a^{\phi(p)} \equiv 1 \bmod p$.

3. *Fermat's theorem:* when p is prime, for every $a \in Z_p^* : a^{(p-1)} \equiv 1 \bmod p$.

   Remark: if the converse of the Fermat's sentence was true, then one could suggest that when we want to test if p is prime, we should check weather, for every $a \in Z_p^*$, it holds that $a^{(p-1)} \equiv 1 \bmod p$. But, the converse of Fermat's theorem is not true and there are numbers, known as the Carmichael numbers, such that $a^{(p-1)} \equiv 1 \bmod p$ for every $a \in Z_p^*$, but p is *not* prime. Therefore, we can not use Fermat's theorem as a primality test.

4. When p is a prime, Z_p^* is a cyclic group of order $p-1$, meaning, there exist an element $g \in Z_p^*$ such that $Z_p^* = \{g^1, g^2, \ldots, g^{p-1}\} = \{g^i : i = 1, 2, \ldots, p-1\}$.

Example: $Z_7^* = \{1,2,3,4,5,6\} = \{3,2,6,4,5,1\} = \{3^1 \bmod 7, 3^2 \bmod 7, 3^3 \bmod 7, 3^4 \bmod p, 3^5 \bmod 7, 3^6 \bmod 7\}$, so 3 is a generator of Z_7^*.

Remark: We can see that for every element $a \in Z_p^*$ and a generator g of it, there exists a unique index i such that: $a \equiv g^i \bmod p$. This i is called the *index* of a with respect to g.

3 Finding a Square Root

3.1 Description of the Problem

Let $a \in Z_p^*$. Given the equation

$$a \equiv x^2 \bmod p$$

is there a solution to this equation?

If a solution x exists such that the above equation is true, a is called *quadratic residue mod p* or a *square mod p*.
If no such solution exists then a is called *quadratic non-residue mod p* or a *non-square mod p*.

What do we know about the number of solutions for such an equation? This depends whether p is a prime or not. If p is prime the following fact is known:

Fact 3.1 *Given a prime p, the equation $a \equiv x^2 \bmod p$ has either no solutions, or exactly 2 solutions: x and $-x$.*

Proof: If the equation has no solutions we are done. Assume there is at least one solution x, then clearly $-x$ is also a solution. We'll show that x and $-x$ are the only possible solutions. Let y be another solution, i.e., $a \equiv y^2 \bmod p$ iff $y^2 \equiv a \bmod p$. Since x is a solution, we also have $a \equiv x^2 \bmod p$ iff $x^2 \equiv a \bmod p$, so we can write

$$\begin{aligned} x^2 &\equiv y^2 \bmod p \\ x^2 - y^2 &\equiv 0 \bmod p \\ (x-y)(x+y) &\equiv 0 \bmod p. \end{aligned}$$

Because of the primality of p, either $y = x$ or $y = p - x = -x$. ■

We will later use this fact to discriminate between *primes* and *non-primes*. So, our next task should be to find a way to know if a given a is a *square mod p*. The following fact will be helpful:

Fact 3.2 *Given a prime p and a generator g for Z_p^*, then $a \in Z_p^*$ is* square mod p *iff the index of a w.r.t. p and g is* even.

Proof:
($\Leftarrow$ even-index imply square:) Let a be an element with an even index, i.e., for some j it holds that $a \equiv g^{2j} \bmod p \equiv (g^j)^2 \bmod p$. This implies that a is a *square mod p*, with a square root g^j.
($\Rightarrow$ square imply even-index:) a is a square-mod-p, so exist $y \in Z_p^*$ s.t. $a \equiv y^2 \bmod p$. Since g is the generator, g generates every member of Z_p^* and specifically y, so for some t, $y \equiv g^t \bmod p$. Plugging the last equation into $a \equiv y^2 \bmod p$ we get: $a \equiv ((g^t \bmod p)^2) \bmod p \equiv g^{2t} \bmod p$. Since the index of a w.r.t g is unique we get that: $i \equiv 2t \bmod (p-1)$, and the index of a is even (because p is prime and hence odd). ■

Example: we saw before that 3 is a generator of Z_7^*, and we note that elements with even index are indeed square-mod-p. For example, notice first that the element 4 has an even index: the index is 4, since $4 \equiv 3^4 \bmod 7$. 4 is a square, with two roots: the obvious 2 and 5, since $4 \equiv 5^2 \bmod 7$. Note that $5 \equiv -2 \bmod 7$ so we see that Fact 3.1 also holds.

Due to the fact that half of the elements are generated with an even index, we can conclude the next corollary:

Corollary 5 *Let p be a prime then half of the elements of Z_p^* are squares-mod-p.*

Remark: keeping in mind our ultimate goal for this class, which is to test whether a number is prime or not, and that our current task is to find if an element of Z_p^* is a square-mod-p, one can suggest the following method to find if a is a square: find the index i of a such that $a = g^i$. If i is even, conclude that a is square-mod-p, else conclude that it is not. This simple idea crushes on the shores of reality due to the fact that finding the index of an element, known as the *discrete log problem* is considered a computationally hard task: that is, no efficient algorithm is known for this task. The next claim will present another way to find out if a is square-mod-p.

Definition 6 *Given a prime p and $a \in Z_p^*$, the Legendre symbol $\left(\frac{a}{p}\right)$ is defined as*

$$\left(\frac{a}{p}\right) = \begin{cases} 1 & \text{a is a square-mod-p} \\ -1 & \text{a is not a square-mod-p} \end{cases}$$

Theorem 7 *Given a prime p and $a \in Z_p^*$,*

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

This is called the *Euler Criteria.*

To prove the claim, first observe that the following fact is correct:

Fact 3.3 *For a prime p, and a* generator *g of Z_p^*, it holds that $g^{\frac{p-1}{2}} = -1$.*

Proof: Look at $\left(g^{\frac{p-1}{2}}\right)^2 = g^{p-1} = 1$ (the last equality holds for every $a \in Z_p^*$). So by Fact 3.1 $g^{\frac{p-1}{2}}$ is either 1 or -1. We'll show that 1 can not be a solution. If $g^{\frac{p-1}{2}} = 1$, we have found an index j smaller then $p-1$ s.t. $g^j = g$. If such index exist g can not generate all the elements in Z_p^*, but since we know that g is a generator, the assumption that $g^{\frac{p-1}{2}} = 1$ can not be true, and we get that $g^{\frac{p-1}{2}} = -1$. ■

Example: we saw that 3 is the generator of Z_7^*. Computing $3^{\frac{7-1}{2}} \bmod 7 = 3^3 \bmod 7 = 27 \bmod 7 \equiv -1$ we see that the fact holds.

We can now prove the Euler Criteria:

Proof (of Theorem 7): If a is a square-mod-p, then by Fact 3.2 a has an even index or in other words $a = g^{2j}$ for some j. Hence, $a^{\frac{p-1}{2}} = g^{2j\frac{p-1}{2}} = g^{j(p-1)} = (g^{p-1})^j = 1^j = 1 = \left(\frac{a}{p}\right)$.

If a is not a square-mod-p, then by Fact 3.2 a has an odd index or in other words $a = g^{2j+1}$ for some j. So,

$$\begin{aligned} a^{\frac{p-1}{2}} &= (g^{2j+1})^{\frac{p-1}{2}} = g^{[j(p-1)+\frac{p-1}{2}]} = g^{j(p-1)}g^{\frac{p-1}{2}} \\ &= (g^{(p-1)^j})(-1) = 1^j(-1) = -1 = \left(\frac{a}{p}\right) \end{aligned}$$

■

3.2 An Algorithm For Finding the Square-Root (mod p)

Now that we have an easy criterion to find out if a is a square-mod-p, we can develop an algorithm for finding the square-root, i.e., an x which satisfy $a \equiv x^2 \bmod p$. This algorithm will give us an efficient way to compute the value of the square-root if it exists.

The input to the Square-Root Algorithm will be prime p and a quadratic-residue $a \in Z_p^*$. The output will be a number x such that $a \equiv x^2 \bmod p$. The algorithm operates differently in two cases, depending on p, one case is easy and the other is more complex.

The two cases For any number p, the value of $p \bmod 4$ is in $\{0, 1, 2, 3\}$. For p that is prime, the possible values are only 1 or 3, since 0 and 2 imply that p is even and hence not a prime. This constitute our two cases:

$$\begin{aligned} &\text{(A)} \quad p \bmod 4 = 3 \\ &\text{(B)} \quad p \bmod 4 = 1 \end{aligned}$$

(A) $p \bmod 4 \equiv 3$. We can write p as $4t+3$ for some t. We assume that a is a square-mod-p so starting from Theorem 7, we develop the following sequence of equivalent conditions:

$$\begin{aligned} \left(\frac{a}{p}\right) &\equiv 1 \bmod p \\ a^{\frac{p-1}{2}} &\equiv 1 \bmod p \\ a^{\frac{4t+3-1}{2}} &\equiv 1 \bmod p \\ a^{2t+1} &\equiv 1 \bmod p \end{aligned}$$

$$\begin{aligned} a^{2t+2} &\equiv a \bmod p \\ a^{2(t+1)} &\equiv a \bmod p \\ \left(a^{t+1}\right)^2 &\equiv a \bmod p. \end{aligned}$$

So we conclude that $x = a^{(t+1)}$ is the square-root of a.

(B) $p \bmod 4 \equiv 1$. Trying to follow the same way as in (A), let us write $p = 4t+1 \Rightarrow a^{\frac{p-1}{2}} = a^{\frac{4t}{2}} = a^{2t} = 1 \bmod p$. But here we hit a wall: multiplying both sides by a as before leaves us with an *odd* power on the left side, and not an even one to get the square-root. Our goal is therefor to have an odd power before the multiplication.

A promising direction to achieve this goal is to write the power of a as $2t = 2^i(2j+1)$. We will first show an "idea" to use this representation to get an odd power in the left side. This "idea" has a problem, but soon we will see how to tackle this problem to get a full solution.

Basic Idea

$$\begin{aligned} a^{\frac{p-1}{2}} &\equiv 1 \bmod p \\ a^{2t} &\equiv 1 \bmod p \\ a^{2^l(2j+1)} &\equiv 1 \bmod p \text{ (take square-root of both sides)} \\ a^{2^{(l-1)}(2j+1)} &\equiv 1 \bmod p \text{ (continue with the square-root until...)} \\ \dots \\ a^{2j+1} &\equiv 1 \bmod p \text{ (multiple both side with } a) \\ a^{2j+2} &\equiv a \bmod p \\ \left(a^{j+1}\right)^2 &\equiv a \bmod p \\ \Rightarrow x = a^{j+1} & \text{ is the square-root of } a. \end{aligned}$$

We have implicitly assumed that taking square roots of $a^2 = 1$ we get $a = 1$. But it may be that $a = -1$. Indeed, what if in one of the square-root operations, the result for the right side is -1 and not 1? In this case we will need an extra "rescue equation". This equation uses a new element b, which is a non-square-mod-p, and hence according to the Euler criteria, $b^{2t} = b^{\frac{p-1}{2}} \equiv -1 \bmod p$. Using this we will manipulate the equation to get an odd power in the left side just before the final step.

The Full Solution We will follow the steps of the basic idea. As long as taking a square-root on both sides results with a 1 in the right side we continue. If the operation results with a -1 (i.e., we reach $a^{2^{(l-1)}(2j+1)} = -1$), we guess a b, and verify that it is indeed non-square with the Euler Criteria. Since $b^{2t} = -1$ we will multiple both side of the 'bad' equation with b^{2t} and get $b^{2t} \cdot a^{2^{(l-1)}(2j+1)} \equiv 1 \bmod p$, and then continue with the sequence of square-root operations. After i times of taking square-root we will get something like:

$$\begin{aligned} b^{\frac{2t}{2^{i_1}}} b^{\frac{2t}{2^{i_2}}} \cdots b^{\frac{2t}{2^{i_k}}} a^{2j+1} &\equiv 1 \bmod p \text{ [for some } i_s\text{'s]} \\ b^{(2^{l_1}+2^{l_2}+\cdots+2^{l_k})(2j+1)} a^{(2j+1)} &\equiv 1 \bmod p \text{ [for } l_s = l - i_s\text{'s, using } 2t = 2^l(2j+1) \text{].} \end{aligned}$$

Observe that the power of b is even. This is true since for each $1 \leq s \leq k$, l_s is positive (at least in the first time of taking square root we take square root from a^{2t} and not from the b term), the power of b is just a sum of the 2^{l_s}'s (multiplied by $2j+1$) and hence is an even number. So writing the power of b as $2D$, for some D, we finally have:

$$
\begin{array}{lcl}
b^{2D}a^{2j+1} & \equiv & 1 \bmod p \text{ (multiple both side with } a\text{)} \\
b^{2D}a^{2(j+1)} & \equiv & a \bmod p \\
\left(b^{D}a^{j+1}\right)^2 & \equiv & a \bmod p \\
\Rightarrow & \text{(the square-root of } a \text{ is:)} & b^{D}a^{j+1}
\end{array}
$$

Algorithm SQRT(a,p)

Input: an odd prime p, and a square-mod-p element a
Output: The element x such that $a \equiv x^2 \bmod p$

A) If $p \bmod 4 = 3$, find t such that $p = 4t+3$.
OUTPUT $a^{t+1} \bmod p$.

B) If $p \bmod 4 = 1$, find t such that $p = 4t+1$.

- B.1) Repeatedly choose uniformly $b \in Z_p^*$, until $b^{\frac{p-1}{2}} \bmod p \equiv -1$
- B.2) initialize $i = 2t$ (the power of a), $k = 0$ (the power of b)
- B.3) REPEAT until i is odd
  - B.3.1) $i \leftarrow \frac{i}{2}$, $k \leftarrow \frac{k}{2}$
  - B.3.2) IF $a^i b^k \equiv -1 \bmod p$ THEN $k \leftarrow k + 2t$
- B.4) OUTPUT $a^{\frac{i+1}{2}} b^{\frac{k}{2}}$

Figure 1: Algorithm SQRT(a,p)

Comment: There is no known efficient deterministic algorithm to perform step B.1 (above), i.e., to find a non-square element in Z_p^*. However, the Extended Riemann Hypothesis (ERH) implies that there is a non-square element in the first $\log^2 p$ numbers of Z_p^*. Hence, this implies a deterministic algorithm, since we can check all the numbers less than $\log^2 p$ and find a non-square.

Correctness of SQRT The correctness of the algorithm follows from the above discussion. Specifically, Stage $B.3.1$ performs the square root operation by dividing the current power of a (which is i) and b (which is k) by 2. As we explained if the square root is -1 we multiply both sides of the equation by -1; this is done in Stage $B.3.2$ by increasing the power of b by $2t$ (which is the same as multiplying by $b^{2t} \equiv -1$).

Analysis of the running time of SQRT It is easy to see that all the deterministic steps run in time which is polynomial with the description of p. What about the random step $B.1$? Since we saw before that half of the elements of Z_p^* are non-square-mod-p, we have a probability of $\frac{1}{2}$ to choose a 'good' (i.e. non-square) element. The *expected* running time is therefore 2 tries for this step, and polynomial-time for all the algorithm.

Comment: the SQRT algorithm described above never fails, but can take in the worst case infinite time, due to a sequence of bad choices in step B.1. It is therefore a *Las-Vegas* type algorithm. The next section describes how it can be turned into a *Monte-Carlo* type algorithm.

3.3 A Monte-Carlo version of the SQRT algorithm

We next provide a monte carlo algorithm $MSQRT(a,p,s)$ for finding square root. The running time is no longer expected polynomial time but rather a polynomial time algorithm. Although, we introduce a failure probability 2^{-s}. The algorithm is different from the last one only in stage B.1. Instead of choosing repeatedly until finding a non-square, we stop after s failures, making sure that the algorithm will always stop, and succeeds with high probability.

The Monte-Carlo version will always run in polynomial-time. For every a and p the algorithm output "FAIL" only when step B.1 did not manage to find a 'good' b in s tries. ¿From Corollary 5 the probability for a failure in one try is $\frac{1}{2}$, the probability to FAIL is $\left(\frac{1}{2}\right)^s = 2^{-s}$.

4 Randomized Algorithm for Primality Testing

The first randomized algorithms for primality testing were given in 1976-77 by Solovay and Strassen and by Miller and Rabin. Both algorithms have only one-side error. We will describe an algorithm given by Silvio Micali. This algorithm is two-sided error but has a simple analyses.

In order to introduce an algorithm that will separate between prime numbers and composite numbers, we need first to find a mathematical property that separates them. This property is needed to be "easy" for being inspected. The property that our algorithm is going to use is the number of solutions to the equation $x^2 \equiv a \bmod n$. By Fact 3.1 we know that if n is a prime number then the number of solutions to the above equation is (0 or) 2. Let $n = \prod_{i=1}^{k} p_i^{\alpha_i}$. We next show that if n is odd and not a prime power (i.e., $k > 1$), then the number of solutions to the above equation is 0 or 2^k.

In order to prove this fact, recall the Chinese remainder theorem (CRT).

Theorem 8 *Let $n = n_1 n_2 \cdots n_k$, where the n_i are pairwise relatively prime. For any sequence of residues $r_1 \in Z_{n_1}, \ldots, r_k \in Z_{n_k}$, there is a unique $r \in Z_n$ such that*

$$r \equiv r_i \bmod n_i \quad (for \quad 1 \le i \le k).$$

Moreover r can be computed in polynomial time.

We next prove the theorem regarding the number of solutions to the equation $x^2 \equiv a \bmod n$.

Theorem 9 *Let $n = \prod_{i=1}^{k} q_i^{\alpha_i}$ be the factorization of an odd number that is not a prime or a prime power. The number of solutions to the equation $x^2 \equiv a \bmod n$ is either 0 or 2^k.*

Proof: Let $n_i = q_i^{\alpha_i}$ and $a_i \equiv a \bmod n_i$. Then by the CRT we know that being equal to $a \bmod n$ is exactly as being equal simultaneously to $a_i \bmod n_i$ for all i's. But Fact 3.1 can be generalized to a prime power such that the equation $x^2 \equiv a_i \bmod n_i$ has either 0 solutions or 2 solutions. If there exists some i with 0 solutions then there is 0 solutions to the equation $x^2 \equiv a \bmod n$ (since if there is any solution to this equation it solves also the equation $x^2 \equiv a_i \bmod n_i$). On the other hand, if every equation $x^2 \equiv a_i \bmod n_i$ has 2 solutions $\{y_i^1, y_i^2\}$ then by using again the CRT there is a unique and different $y \in Z_n$ that corresponds to any one of the 2^k choices of simultaneous solutions for the equations (choosing an array of length k each cell with two choices). ■

We next present the algorithm Primality-Test. The algorithm first chooses uniformly at random solution to the equation $x^2 \equiv a \bmod p$. This is done by choosing x at random and then squaring it. Then we ask algorithm $MSQRT(a,p,s)$ to find a sqrt to a. If p is prime then the only solutions are $\{x, -x\}$ and, with probability $1 - 2^{-s}$, algorithm $MSQRT(a,p,s)$ finds this solution. But if p is not a prime then even if algorithm $MSQRT(a,p,s)$ returns a square root with good probability, it may happen to be *not* in $\{x, -x\}$, but rather be another solution (one of the other $2^k - 2$ possible solutions). The algorithm Primality-Test is presented formally in Figure 2.

Algorithm Primality-Test.
Input: a number p and a safety parameter s.
Output: "Prime" or "Composite".

1. If p is either even or a prime power return "Prime".
2. Choose $x \in Z_p^*$ uniformly at random.
3. If $gcd(x,p) \neq 1$ return "Composite".
4. Compute $a = x^2 \bmod p$.
5. $val \leftarrow MSQRT(a,p,s)$.
6. If val equals "FAIL" or $val \notin \{x, -x\}$ or $val^2 \neq a \bmod p$ then return "Composite",
7. Else (i.e., $val \in \{x, -x\}$) return "Prime".

Figure 2: Algorithm Primality-Test

Theorem 10 *Algorithm* Primality-Test(p,s) *is a monte carlo algorithm that runs in polynomial time and distinguish between a prime number and a composite number. If p is prime then the algorithm outputs "prime" with probability at least $1 - 2^{-s}$ and if p is composite then the algorithm outputs "composite" with probability at least $\frac{1}{2}$.*

Proof: We first analyze the running time of the algorithm. The first step is implemented in polynomial time, since one can check if a number p is a power just by trying all the possible powers $k \leq \log p$ and finding a possible solution on the integers by a simple binary search (each time dropping half of the possible integer solutions). Then we use Euclid algorithm and algorithm $MSQRT(a,p,s)$, which are both polynomial time algorithms.

We next prove two claims regarding the two sided errors.

Claim 4.1 *If p is a prime number then the probability that algorithm* Primality-Test(p,s) *output "Composite" is less than 2^{-s}.*

Proof: If p is a prime number than algorithm $MSQRT(a,p,s)$ returns the unique solutions $\{x,-x\}$ with probability at least $1-2^{-s}$. ∎

Claim 4.2 *If p is composite then the probability that algorithm* Primality-Test(p,s) *outputs "Prime" is at most $\frac{1}{2}$.*

Proof: If p is composite then we can guarantee nothing on the output of $MSQRT(a,s)$. But even if this algorithm outputs a square root of a then by Theorem 9 it is one of 2^k possible solutions, where k is the number of primes that divides p. Since $k \geq 2$, the probability that the output of $MSQRT$ is in $\{x,-x\}$ is at most $\frac{2}{2^k} \leq \frac{1}{2}$. It follows that if p is composite then the algorithm outputs "prime" with probability at most $\frac{1}{2}$. ∎

Combining the two claims, the theorem follows. ∎

5 Short proof for prime numbers

Let **PRIME** be the set of all prime numbers and let **COMPOSITE** denote the complement set. It is easy to see that **COMPOSITE**$\in \mathcal{NP}$, since the witness for a number n is simply two integers p and q such that their product is n (and none is 1). Computing a product of two numbers is polynomial in the *length* of the numbers. We emphasize that checking if the $\mathcal{NP}$ witness is good must be polynomial in the length of n which is $\log n$. Our goal in this section is to show that there exists short proofs also for **PRIME**.

In first glance this result is somehow surprising since **PRIME** is a set that is a co$\mathcal{NP}$ kind (i.e., for p in the set one must prove that for **every** $x < p$, x does not divide p). However, Pratt discovered a short proof for **PRIME**. Since **COMPOSITE** is also in $\mathcal{NP}$ this means that **PRIME** $\in \mathcal{NP} \cap$ co$\mathcal{NP}$ hence it is believed that **PRIME** is not $\mathcal{NP}$-COMPLETE.

Again we are looking for a mathematical property that separates the prime numbers from the composite. By Corollary 2.1 we know that Z_p^* has $p-1$ elements if p is a prime number, otherwise it has less elements. The first idea to use this characterization (of the primes) is to use the fact that Z_p^* has a generator g. Given such a g we just have to check that the order of g is $p-1$. It seems that we did nothing because this can be checked by looking at the set $\{g^1, g^2, \ldots, g^{p-1}\}$ and verifying that the only power that equals 1 is $p-1$. But the size of this set is exponential in the length of p. In order to solve this problem we use Fact 5.1. From this fact we conclude that it suffices to check a set of size at most $O(\log p)$ (the number of distinct primes that divides p) in order to guarantee that g is a generator of Z_p^* and that the order of g is $p-1$.

Fact 5.1 *Let p be a prime then g is a generator of Z_p^* iff for every prime q that divides $p-1$ it holds that $g^{\frac{p-1}{q}} \neq 1$.*

Proof: If g is a generator and p is prime clearly its order is $p-1$ hence any power of g less than $p-1$ does not equals 1. Conversely, if g is not a generator then the set $\{g^1, g^2, \ldots, g^{ord(g)}\}$ is a subgroup of Z_p^* and thus divides $p-1$. But if $ord(g) < p-1$ and divides $p-1$ it must also divides $\frac{p-1}{q}$ for some prime q. Hence we conclude that $g^{\frac{p-1}{q}} \equiv 1 \bmod p$. ∎

But given the prime factorization of $p-1 = \prod_{i=1}^{k} q_i^{\alpha_i}$; how do we know that every q_i is itself a prime? this is done recursively.

Theorem 11 *Let p be a prime number and $p-1=\prod_{i=1}^{k} q_i^{\alpha_i}$. The following recursive witness for p,*

$$W(p) = g \circ (q_1,\alpha_1) \circ (q_2,\alpha_2) \circ \cdots \circ (q_k,\alpha_k) \circ W(q_1) \circ W(q_2) \cdots W(q_k) \ ,$$

is of polynomial size in the length of p and can be verified in polynomial time, hence $PRIME \in \mathcal{NP}$.

Proof: In order to check the witness $W(p)$, for the integer p, we first verify that indeed $p-1=\prod_{i=1}^{k} q_i^{\alpha_i}$. Then we check that $g^{\frac{p-1}{q_i}} \neq 1 \bmod p$, for every $1 \leq i \leq k$. Finally, we continue by checking recursively that indeed each q_i is a prime by checking all the $W(q_i)$'s. We have already explained why $W(p)$ is a witness for p.

We next prove that it is of length polynomial in the length of p. Open the recursion in a tree fashion and look at the i'th level of the tree. Notice (can be checked recursively) that the product of all the q's in the i'th level is always less than p. Hence their total length is less than twice the length of p. If we add the generators of the i'th level and the α's we get no more than $4\log p$. The number of levels is less than $\log p$ because each time there is at least a division by 2. Multiplying the number of levels with the size of each level completes the claim about the size of the witness.

The time it takes to verify a witness includes mainly checking that $g^{\frac{p-1}{q_i}} \neq 1$ for every $1 \leq i \leq k$. Since taking powers is logarithmic in the exponent, we can conclude that each "piece" of the witness needs $O(poly(\log p))$ time to verify, all together polynomial time in $O(\log p)$. ■