University of Tokyo: Advanced Algorithms **Summer 2010**

Lecture 6 — 27 May

Lecturer: François Le Gall *Scribe: Baljak Valentina*

As opposed to prime factorization, primality testing is determining whether a given number is a prime, without necessarily computing its factorization. This lecture will study in details Solovay-Strassen algorithm for primality testing, discuss random primes generation and present one of its applications to cryptography.

Let's keep in mind definitions from the previous lecture:

Legendre symbol: Let p be an odd prime, and a be any integer. Then the Legendre symbol is defined as $\left(\frac{a}{p}\right) = a^{(p-1)/2} \bmod p$, where we treat $p-1$ as -1.

Jacobi symbol: Let $n > 1$ be an odd integer with factorization $n = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k}$. Then, for any integer a, the Jacobi symbol is defined as $\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{e_1} \left(\frac{a}{p_2}\right)^{e_2} \dots \left(\frac{a}{p_k}\right)^{e_k}$. The Jacobi symbol $\left(\frac{a}{1}\right)$ is defined to be 1 for any integer a.

6.1 Solovay-Strassen Algorithm

The Solovay–Strassen primality test is a probabilistic test to determine if a number is composite or probably prime, and has a great importance in showing the practical feasibility of the RSA cryptosystem, which will be explained further in this lecture.

Definition 1. *For odd $n \geqslant 3$, we define $E(n) = \{a \in \mathbb{Z}_n^* | \left(\frac{a}{n}\right) = a^{(n-1)/2} \bmod n\}$.*

We will use the following lemma.

Lemma 6.1. *For odd $n \geqslant 3$, n is prime if and only if $E(n) = \mathbb{Z}_n^*$.*

For the proof of the lemma, refer to the book Randomized Algorithms [1], Lemma 14.30.

We now describe Solovay–Strassen algorithm, which takes as input an odd integer $n \geqslant 3$.

Algorithm 1 Solovay-Strassen(n) $[n \geq 3 \text{ odd}]$

Take a at random in $\{1, 2, \ldots, n-1\}$
if $gcd(a, n) \neq 1$ **then**
  return("*composite*")
else if $\left(\frac{a}{n}\right) \neq a^{(n-1)/2} \bmod n$ **then**
    return("*composite*")
  else
    return("*prime*")
  end if
end if

Running time: $O((\log n)^3)$. This follows from running times of separate parts of the algorithm: finding gcd, computing of Jacobi symbol, and finally computing powers of a. Respectively, $O((\log n)^2) + O((\log n)^2) + O((\log n)^3)$.

Theorem 6.2. *If n is an odd prime, and $a \in \{1, \ldots, n-1\}$, the probability that the algorithm returns "prime" is*

$$\Pr_{a \in \{1,\ldots,n-1\}}[Solovay - Strassen(n) = \text{"}prime\text{"}] = 1.$$

If n is an odd composite, the probability that algorithm returns "composite" is

$$\Pr_{a \in \{1,\ldots,n-1\}}[Solovay - Strassen(n) = \text{"}composite\text{"}] \geqslant \frac{1}{2}.$$

Proof: If n is an odd prime, then the algorithm will obviously always output "prime".

Let us now prove the second part of the theorem. Assume that n is an odd composite. We will show that the probability of the algorithm returning "prime" is $\leq \frac{1}{2}$.

$$\Pr_{a \in \{1,\ldots,n-1\}}[Solovay - Strassen(n) = \text{"}prime\text{"}] =$$

$$\Pr_{a \in \{1,\ldots,n-1\}}[\{gcd(a, n) = 1\} \wedge \{\left(\tfrac{a}{n}\right) = a^{(n-1)/2} \bmod n\}] = \frac{E(n)}{n-1}$$

From Lemma 6.1 it follows that $E(n) \neq \mathbb{Z}_n^*$. Now it is easy to show that $E(n)$ is a **subgroup** of the multiplicative group $\mathbb{Z}_n^*$:

$$a, b \in E(n) \Rightarrow (ab \bmod n) \in E(n)$$
$$a \in E(n) \Rightarrow a^{-1} \in E(n).$$

$E(n)$ is thus a proper subgroup of $\mathbb{Z}_n^*$ and, from elementary group theory, we conclude that

$$|E(n)| \leqslant \frac{|\mathbb{Z}_n^*|}{2} \leqslant \frac{n-1}{2}.$$

Thus $\Pr_{a \in \{1,\ldots,n-1\}}[Solovay - Strassen(n) = \text{"}prime\text{"}] \leq \frac{1}{2}$. □

Remark: we can give an alternative proof of the fact that $|E(n)| \leqslant |\mathbb{Z}_n^*|/2$ when n is an odd composite without using group theory: We know that $E(n) \neq \mathbb{Z}_n^*$. So there has to exist an element $b \in \mathbb{Z}_n^* \setminus E(n)$. Define the set $A(n) = \{ab | a \in E(n)\}$ (see Figure 6.1). It is easy to show that $|A(n)| = |E(n)|$ and $A(n) \cap E(n) = \varnothing$. Thus $|E(n)| \leqslant |\mathbb{Z}_n^*|/2$.

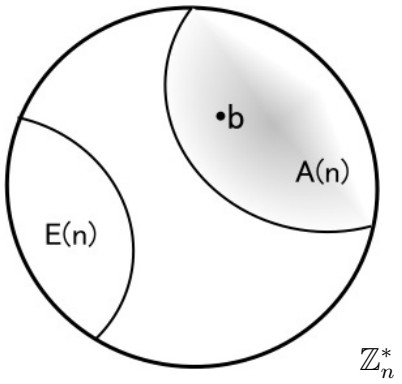


Figure 6.1.

Example Let's take $n = 15$. Notice that $a = 3, 5, 6, 9, 10, 12$ will give a "composite" output. For the other values of a:

a	$\left(\frac{a}{15}\right)$	$\mathbf{a}^7 \bmod 15$
1	1	1
2	1	8
4	1	4
7	-1	13
8	1	2
11	-1	11
13	-1	7
14	-1	14

The algorithm will output "prime" only for $a = 1$ and $a = 14$. Then the probability of algorithm returning composite in this case is:

$$\Pr_{a \in \{1,\ldots,n-1\}}[Solovay - Strassen(15) = \text{“}composite\text{”}] = \frac{12}{14}.$$

Remarks

1. The error probability can be decreased to $\frac{1}{2^k}$ by repeating Solovay-Strassen algorithm k times.

2. The Rabin-Miller test, another randomized algorithm used for the same purpose, also has basically the same complexity as Solovay-Strassen algorithm.

6.2 Generating Random Primes

Random primes are widely used in cryptosystems such as the RSA cryptosystem we will consider later.

For any positive integer x, define $\Pi(x)$ as the number of primes $p \leqslant x$. From the **prime number theorem** we know that

$$\frac{x}{\ln(x)} < \Pi(x) < 1.26 \cdot \frac{x}{\ln(x)} \text{ for } x > 17.$$

As a consequence, suppose that we want to generate a random prime in $\{m+1, \ldots, 2m\}$. The number of primes in this set is:

$$\Pi(2m) - \Pi(m) > \frac{2m}{\ln(2m)} - 1.26 \cdot \frac{m}{\ln(m)} = \Omega\left(\frac{m}{\log(m)}\right),$$

where the $\Omega(\cdot)$ notations means that there exists a constant $c > 0$ such that, for m sufficiently large,

$$\frac{2m}{\ln(2m)} - 1.26 \cdot \frac{m}{\ln(m)} \geq c \cdot \left(\frac{m}{\log(m)}\right).$$

The probability that a random number in the given interval is prime is thus $\Omega\left(\frac{1}{\log(m)}\right)$ and it enables one to design algorithms for generating random primes, such as the following one.

Algorithm 2 Random(m, k)

loop
  Take a at random in $\{m+1, \ldots, 2m\}$
  for $i = 1$ to k **do**
    $Solovay - Strassen(a)$
  end for
  if $Solovay - Strassen(a) =$"prime" k times **then**
    return a
  end if
end loop

Theorem 6.3. *The expected number of times the loop is run is $O(\log(m))$. The expected running time is $O(k(\log m)^4)$. When the algorithm stops, the returned number is a prime $\{m+1,\ldots,2m\}$ with a probability $\geqslant 1-\frac{1}{2^k}$.*

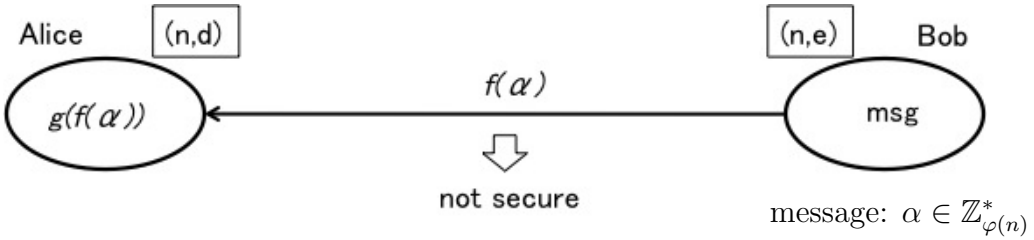
Note: In order to obtain the upper bound on the expected number of time the loop is run, let's keep in mind the flipping of a biased coin. If the probability that a head appears is p and that a tail appears is $1-p$, then the expected number of flips until obtaining a head is $\frac{1}{p}$.

The presented algorithm can be easily improved with better bounds and more efficient tests.

6.3 RSA Cryptosystem

The RSA cryptosystem is a public-key cryptosystem created by Rivest, Shamir and Adleman in 1978.

Scenario. Suppose there are two players, Alice and Bob. Bob wants to send a message to Alice. One solution is to use a secret key, but, in order to use this key, it must be shared, and this is a problem if it needs to be sent, since the communication is insecure. If we use RSA, encryption and decryption keys are different and this problem does not happen.



The idea

1. Alice takes two big random primes, p and q (≈ 500 bits long).
2. Alice sets $n = pq$ and takes a random $e \in \mathbb{Z}^*_{\varphi(n)}$, where $\varphi(n) = \varphi(p)\varphi(q) = (p-1)(q-1)$, Euler's function.
3. Alice computes $d \in \mathbb{Z}^*_{\varphi(n)}$ such that $d \cdot e \bmod \varphi(n) = 1$.
4. The public key (n, e) is announced and can be used for encryption, and the private key (n, d) is used for decryption and it is known only to Alice.

We suppose that Bob want to send a message $\alpha \in \mathbb{Z}^*_{\varphi(n)}$ to Alice. The encryption and decryption functions are as follows.

Encryption function $f: \begin{array}{c} \mathbb{Z}^*_n \to \mathbb{Z}^*_n \\ x \to x^e \bmod n \end{array}$

Decryption function $g: \begin{array}{c} \mathbb{Z}^*_n \to \mathbb{Z}^*_n \\ y \to y^d \bmod n \end{array}$

It is important to note that $g(f(\alpha)) = \alpha$, as we now show.

Proof: By definition, $f(\alpha) = \alpha^e \bmod n$. Thus

$$g(f(\alpha)) = \alpha^{ed} \bmod n = \alpha^{1+c\varphi(n)} \bmod n$$

for some integer c. Remember Euler's theorem: $\alpha^{\varphi(n)} \bmod n = 1$ for all $\alpha \in \mathbb{Z}^*_{\varphi(n)}$. We conclude that

$\boxed{g(f(\alpha)) = \alpha \bmod n = \alpha.}$ □

Concluding notes: We want to argue that this protocol is secure. This has to be based on some assumptions. One possible attack is to compute $\varphi(n)$ from n and then to compute d. The question is, **can we compute** $\varphi(n)$ **from** n?

It can be shown that computing this function is as hard as factoring n when $n = p \cdot q$, and it is generally believed that it can not be achieved in a reasonable amount of time (more precisely, in polynomial time) with the current level of computation technology. RSA is then believed to be secure. However, notice that the same task is easy for quantum computers: using Shor's quantum algorithm one can find prime factors of an integer in time polynomial [2]. Thus, if a large-scale quantum computer can be built, the RSA cryptosystem will be broken.

Bibliography

[1] R. Motwani and P. Raghavan, *Randomized algorithms*. Cambridge University Press, 1995.

[2] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Journal on Computing*, vol. 26, no. 5, pp. 1484–1509, 1997.