Math 261C: Randomized Algorithms

Lecture topic: Factorization, and RSA Public Key System

Lecturer: Sam Buss
Scribe notes by: Sittipong (Kuang) Thamrongpairoj
Date: April 28, 2014

1. Factorization Algorithm

In the last lecture, we gave an algorithm of how to find a square root of a quadratic residue mod prime p. In a homework problem, we can construct an algorithm similar to the one given in the lecture to find a square root mod prime power p^k. To completely solve this problem, we try to answer the question of how to find a square root mod composite number n. It turns out that this question is as hard as finding a nontrivial factor of n.

Given a positive integer n, we want to find an algorithm that gives a nontrivial factor for n. Here is a motivation of how to factor n: if we somehow could find two integers

- x, y such that $x^2 \equiv y^2 \mod n$
- $x \not\equiv \pm y \mod n$

that is x is a "non-trivial square root of y^2, then we know that $(x-y)(x+y) \not\equiv 0$, while both $x-y$ and $x+y$ are not $0 \mod n$. It can be easily concluded that $\gcd(n, x-y) \neq 1, n$, so $\gcd(n, x-y)$ is a non-trivial factor for n.

Suppose we have an algorithm $A(n, u)$ computing a square root of u. We construct an algorithm finding a non-trivial factor for n.

Input: n
Output: non-trivial factor for n
repeat
  Choose $x \in \{1, 2, \ldots n-1\}$ randomly
  if $\gcd(x, n) \neq 1$ **then**
    output $\gcd(x, n)$
  end
  Let $y = A(x^2, n)$
  if $\gcd(x-y, n) \neq 1, n$ **then**
    output $\gcd(x-y, n)$
  else
    Repeat loop;
  end
until *there is an output*;

The reason that this algorithm will sucuessfully produce a factor for n is that, if n has l prime factors, then any quadratic residue $u \mod n$ has 2^l square roots. Thus, the probability that $A(x^2, u)$ will evaluate $\pm x$ is $\frac{1}{2^l} \leq \frac{1}{2}$. Thus, the expected number of iterations of the loop needed until a nontrivial factor of n is produced is bounded by $\frac{1}{1-2^{-l}} \leq 2$.

2. RSA Public Key System

Consider the following situation: Alice and Bob want to communicate secretly through a public communication system. Is there a way Alice and Bob can send an encrypted public message, such that only them two understand the message? The RSA Public Key System allows them to do that.

Firstly, Alice chooses primes p, q randomly, and let $n = pq$. Alice computes $\phi(n) = (p-1)(q-1) = n - p - q + 1$. Then she chooses $k \in \mathbb{Z}^*_{\phi(n)}$, and computes $l = k^{-1} \mod \phi(n)$, meaning, finds l such that $lk = 1 \mod \phi(n)$. Then, Alice publishss n, k, and keeps everything else secret. n, k are known as public keys, while l is known as a private key.

Bob receives keys n, k. If Bob wishes to send message $M < n$ to Alice, he computes and publishes $M^k \mod n$. Alice receives M^k, then computes $(M^k)^l \equiv M^{kl} \equiv M \mod n$.

The reason that made the this algorithm works is that it is hard to compute l without knowing the value of $\phi(n)$, and it is hard to compute $\phi(n)$ without knowing the factorization of $n = pq$. Then a question arises, how hard is it to figure out l? Or how hard is it to compute y^l for any given y? In other words, how hard is it to find a k^{th} root of a number mod n. The next theorem says, if there is an algorithm for computing y^l correctly for any small subset of $\mathbb{Z}^*_n$, then there is an algorithm, based on the first algorithm, which computes y^l correctly for all $y \in \mathbb{Z}^*_n$.

Theorem 1. *Fix values for n, p, q, k, l. If there is a randomized algorithm B_1, a subset S of $\mathbb{Z}^*_n$, $|S| \geq \epsilon|\mathbb{Z}^*_n|$ such that $B_1(y) = y^l$ for all $y \in S$, Then there is a randomized algorithm B_2 such that $B_2(y) = y^l$ for all $y \in \mathbb{Z}^*_n$.*

B_2's expected runtime is $O(\frac{1}{\epsilon} \cdot$ (Expected runtime of B_1))

Proof. Suppose we have an algorithm B_1 which output y^l correctly for $y \in S$. Given below is an algorithm for B_2

Input: y, n, k
Output: k^{th} root of $y \mod n$
repeat
  Choose $z \in \mathbb{Z}_n^*$ randomly
  Let $y' = yz^k$
  Let $x = B_1(y)$
  if $x^k = y'$ *(means that B_1 computes correctly at y')* **then**
    output xz^{-1}
  else
    Repeat loop;
  end
until *there is an output*;

If B_1 computes correctly at y', namely $x^k = y'$, then the algorithm works because $(xz^{-1})^k \equiv x^k z^{-k} \equiv y' z^{-k} \equiv yz^k z^{-k} \equiv y \mod n$, so xz^{-1} is indeed the k^{th} root of y.

It remains to prove that the algorithm will eventually works, and the expected runtime is as described. Here, note that the map $z \mapsto yz^k$ is a bijection. Therefore, by uniformly randomly choosing z, y' is uniformly distributed from $\mathbb{Z}_n^*$. The probability that B_1 will computes y' correctly is equal to the probability that y' is in S, which is ϵ. Thus, the expected number of times we random z is $\frac{1}{\epsilon}$, which completes the proof. □

The above construction is an example of the technique known as "Hardness Amplification". We expressed the construction in contrapositve form as an "Easiness Amplification"; namely, if there is a large number of inputs where the algorithm B_1 succeeds, then there is another algorithm B_2 which succeeds on all inputs.

3. Factorization of n, given $\phi(n)$

Next, we prove another theorem concerning factorization of an integer. Suppose we want to factorize an integer n, but we were also given a value of $\phi(n)$. In this case, we can find a non-trivial factor for n quickly.

Theorem 2. *There is a randomized algorithm, expected polynomial runtime, which given composite n and the value of $\phi(n)$, outputs nontrivial factor for n.*

Proof. Without loss of generality, assume that n is odd. (Otherwise, output 2). Recall, if $n = \prod_p p^{k_p}$, then $\phi(n) = \prod_p p^{k_p - 1}(p-1)$. Then, if n is not square free, meaning that there is a prime factor p such that $k_p > 1$, then $p | \gcd(n, \phi(n))$, so just output $\gcd(n, \phi(n))$

Otherwise, we may assume that $n = \prod_p p$ product of distinct primes.

Input: $n, \phi(n)$, n is square-free
Output: non-trivial factor for n
repeat
  Choose $x \in \{1, 2, \ldots, n-1\}$ randomly
  if $\gcd(x, n) \neq 1$ **then**
    output x
  end
  for $t = 1, 2, \ldots, \log(\phi(n))$ **do**
    if $2^t | \phi(n)$ **then**
      $y_t = x^{\frac{\phi(n)}{2^t}}$
      if $\gcd(y_t - 1, n) \neq 1$ **then**
        output $\gcd(y_t - 1, n)$
      end
    end
  end
until *there is an output*;

Here is a reason why this algorithm works. Consider $x, y_l \mod p_i$ for each $i = 1, 2, \ldots, l$. We write

$$\phi(n) = (p_i - 1)2^{t_i} T_i$$

where T_i is an odd integer. Without lost of generality, assume t_1 is the largest integer among t_i's. For random x, with probability of at least $\frac{1}{4}$, we have

(1) x is a quadratic residue $\mod p_1$

(2) x is not a quadratic residue $\mod p_2$

Thus, consider at $t = t_1 + 1$, $y_t = y_{t_1+1} = x^{\frac{\phi(n)}{2^{t_1+1}}}$. Since x is a quadratic residue $\mod p_1$, $x^{\frac{p-1}{2}} = 1 \mod p_1$, which implies $y_t = 1 \mod p_1$. However, since $t_1 \geq t_2$ and x is not a quadratic residue $\mod p_2$, $y_t \neq 1 \mod p_2$.

Therefore, $y_t - 1$ has a factor of p_1, but not p_2, so $\gcd(y_t - 1, n) \neq 1, n$. □