Lecture Notes 10

50 Tests of Compositeness: Formal Definition

Formally, a *test of compositeness* is a set $T = \{\tau_1, \ldots, \tau_s\}$, where $\tau_i : \mathbf{Z} \to \{\mathsf{true}, \mathsf{false}\}$ has the property that

$$\tau_a(n) = \mathsf{true} \Rightarrow \ n \text{ is composite.}$$

If $\tau_a(n) = \mathsf{true}$, we say that $\tau_a(n)$ *succeeds*, and a is a *witness* to the compositeness of n. If $\tau_a(n) = \mathsf{false}$, then the test *fails* and gives no information about the compositeness of n. Clearly, if n is prime, then all τ_a fail on n, but if n is composite, then $\tau_a(n)$ may either succeed or fail.

A test of compositeness T is *useful* if there is a feasible algorithm $T(n, a)$ that computes $\tau_a(n)$, and for every composite number n, a fraction $c > 0$ of the tests succeed on n. Suppose for simplicity that $c = 1/2$ and one applies 100 randomly-chosen tests to n. If any of them succeeds, we have a proof that n is composite. If all fail, we don't know whether or not n is prime or composite. But what we do know is that if n is composite, the probability that all 100 tests fail is only $1/2^{100}$.

In practice, what we do to choose RSA primes p and q is to choose numbers at random and apply some fixed number of randomly-chosen tests to each candidate,[1] rejecting the candidate if it proves to be composite. We keep the candidate (and assume it to be prime) if all of the tests for compositeness fail. We never know whether or not our resulting numbers p and q really are prime, but we can adjust the parameters to reduce the probability to an acceptable level that we will end up a number p or q that is not prime (and hence that we have unknowingly generated a bad RSA key).

51 Example Tests of Compositeness

Here are two examples of tests for compositeness.

1. Let $\delta_a(n) = (2 \le a \le n-1 \text{ and } a|n)$. Test δ_a succeeds on n if a is a proper divisor of n, which indeed implies that n is composite. Thus, $\{\delta_a\}_{a \in \mathbf{Z}}$ is a valid test of compositeness. Unfortunately, it isn't very useful in a probabilistic primality algorithm since the number of tests that succeed when n is composite are too small. For example, if $n = pq$ for p, q prime, then the *only* tests that succeed are δ_p and δ_q.

2. Let $\zeta_a(n) = (2 \le a \le n-1 \text{ and } a^{n-1} \not\equiv 1 \pmod{n})$. By Fermat's theorem, if p is prime and $\gcd(a, p) = 1$, then $a^{p-1} \equiv 1 \pmod{p}$. Hence, if $\zeta_a(n)$ succeeds, it must be the case that n is *not* prime. This shows that $\{\zeta_a\}_{a \in \mathbf{Z}}$ is a valid test of compositeness. For this test to be adequate for a probabilistic primality algorithm, we would need to know that for all composite numbers n, a significant fraction of the tests ζ_a succeed on n. Unfortunately, there are certain compositeness numbers n called *pseudoprimes* for which all of the tests ζ_a fail. Such n are fairly rare, but they do exist. The ζ_a tests are unable to distinguish pseudoprimes from true primes, so they are not adequate for testing primality.

[1] This is what Algorithm P_2 does.

We will return to this topic later when we have developed sufficient number theory to present tests of compositeness that do have the properties needed to make them useful in probabilistic primality algorithms.

52 Chinese Remainder Theorem

We now return to a basic result of number theory that will be used later.

Let $n_1, n_2, \ldots, n_k$ be positive *pairwise relatively prime* positive integers[2], let $n = \prod_{i=1}^{k} n_i$, and let $a_i \in \mathbf{Z}_i$ for $i = 1, \ldots, k$. Consider the system of congruence equations with unknown x:

$$\begin{array}{rcl} x & \equiv & a_1 \pmod{n_1} \\ x & \equiv & a_2 \pmod{n_2} \\ & \vdots & \\ x & \equiv & a_k \pmod{n_k} \end{array} \tag{1}$$

The *Chinese Remainder Theorem* says that (1) has a unique solution in $\mathbf{Z}_n$.

To solve for x, let

$$N_i = n/n_i = \underbrace{n_1 n_2 \ldots n_{i-1}} \cdot \underbrace{n_{i+1} \ldots n_k},$$

and compute $M_i = N_i^{-1} \bmod n_i$, for $1 \leq i \leq k$. Note that $N_i^{-1} \pmod{n_i}$ exists since $\gcd(N_i, n_i) = 1$ by the pairwise relatively prime condition. We can compute N_i^{-1} using the methods of section 46 (lecture notes 9). Now let

$$x = (\sum_{i=1}^{k} a_i M_i N_i) \bmod n \tag{2}$$

If $j \neq i$, then $M_j N_j \equiv 0 \pmod{n_i}$ since $n_i | N_j$. On the other hand, $M_i N_i \equiv 1 \pmod{n_i}$ by definition of M_i. Hence,

$$x \equiv \sum_{i=1}^{k} a_i M_i N_i \equiv \underbrace{0a_1 + \ldots + 0a_{i-1}} + 1a_i + \underbrace{0a_{i+1} \ldots 0a_k} \equiv a_i \pmod{n_i} \tag{3}$$

for all $1 \leq i \leq k$, establishing that (2) is a solution of (1).

To see that the solution is unique in $\mathbf{Z}_n$, let χ be the mapping $x \mapsto (x \bmod n_1, \ldots, x \bmod n_k)$. χ is a surjection[3] from $\mathbf{Z}_n$ to $\mathbf{Z}_{n_1} \times \ldots \times \mathbf{Z}_{n_k}$ since we have just shown for all $(a_1, \ldots, a_k) \in \mathbf{Z}_{n_1} \times \ldots \times \mathbf{Z}_{n_k}$ that there exists $x \in Z_n$ such that $\chi(x) = (a_1, \ldots, a_k)$. Since also $|\mathbf{Z}_n| = |\mathbf{Z}_{n_1} \times \ldots \times \mathbf{Z}_{n_k}|$, χ is a bijection, and (1) has only one solution in $\mathbf{Z}_n$.

53 Homomorphic property of χ

The bijection χ is interesting in its own right, for it establishes a one-to-one correspondence between members of $\mathbf{Z}_n$ and k-tuples $(a_1, \ldots, a_k)$ in $\mathbf{Z}_{n_1} \times \ldots \times \mathbf{Z}_{n_k}$. This lets us reason about and compute with k-tuples and then translate the results back to $\mathbf{Z}_n$.

The *homomorphic property* of χ means that performing an arithmetic operation on $x \in \mathbf{Z}_n$ corresponds to performing the similar operation on each of the components of $\chi(x)$. More precisely, let

[2] This means that $\gcd(n_i, n_j) = 1$ for all $1 \leq i < j \leq k$.

[3] A *surjection* is an onto function.

$\odot$ be one of the arithmetic operations $+$, $-$, or $\times$. If $\chi(x) = (a_1, \ldots, a_k)$ and $\chi(y) = (b_1, \ldots, b_k)$, then

$$\chi((x \odot y) \bmod n) = ((a_1 \odot b_1) \bmod n_1,\ \ldots,\ (a_k \odot b_k) \bmod n_k). \tag{4}$$

In other words, if one first performs $z = (x \odot y) \bmod n$ and then computes $z \bmod n_i$, the result is the same as if one instead first computed $a_i = (x \bmod n_i)$ and $b_i = (y \bmod n_i)$ and then performed $(a_i \odot b_i) \bmod n_i$. This relies on the fact that $(z \bmod n) \bmod n_i = z \bmod n_i$, which holds because $n_i \,|\, n$.

54 RSA Decryption Works for All of Z_n

In section 42 (lecture notes 8), we showed that RSA decryption works when $m, c \in \mathbf{Z}_n^*$ but omitted the proof that it actually works for all $m, c \in \mathbf{Z}_n$. We now use the Chinese Remainder Theorem to supply this missing piece.

Let $n = pq$ be an RSA modulus, p, q distinct primes, and let e and d be the RSA encryption and decryption exponents, respectively. We show $m^{ed} \equiv m \pmod{n}$ for all $m \in \mathbf{Z}_n$.

Define $a = (m \bmod p)$ and $b = (m \bmod q)$, so

$$\begin{aligned} m &\equiv a \pmod{p} \\ m &\equiv b \pmod{q} \end{aligned} \tag{5}$$

Raising both sides to the power ed gives

$$\begin{aligned} m^{ed} &\equiv a^{ed} \pmod{p} \\ m^{ed} &\equiv b^{ed} \pmod{q} \end{aligned} \tag{6}$$

We now argue that $a^{ed} \equiv a \pmod{p}$. If $a \equiv 0 \pmod{p}$, then obviously $a^{ed} \equiv 0 \equiv a \pmod{p}$. If $a \not\equiv 0 \pmod{p}$, then $\gcd(a, p) = 1$ since p is prime, so $a \in \mathbf{Z}_p^*$. By Euler's theorem,

$$a^{\phi(p)} \equiv 1 \pmod{p}$$

Since $ed \equiv 1 \pmod{\phi(n)}$, we have $ed = 1 + u\phi(n) = 1 + u\phi(p)\phi(q)$ for some integer u. Hence,

$$a^{ed} \equiv a^{1+u\phi(p)\phi(q)} \equiv a \cdot \left(a^{\phi(p)}\right)^{u\phi(q)} \equiv a \cdot 1^{u\phi(q)} \equiv a \pmod{p} \tag{7}$$

Similarly,

$$b^{ed} \equiv b \pmod{q} \tag{8}$$

Combining the pair (6) with (7) and (8) yields

$$\begin{aligned} m^{ed} &\equiv a \pmod{p} \\ m^{ed} &\equiv b \pmod{q} \end{aligned}$$

Thus, m^{ed} is a solution to the system of equations

$$\begin{aligned} x &\equiv a \pmod{p} \\ x &\equiv b \pmod{q} \end{aligned} \tag{9}$$

From (5), m is also a solution of (9). By the Chinese Remainder Theorem, the solution to (9) is unique modulo n, so $m^{ed} \equiv m \pmod{n}$ as desired.

55 RSA Security

Several possible attacks on RSA are discussed below and their relative computational difficulties discussed.

55.1 Factoring n

The security of RSA depends on the computational difficulty of several different problems, corresponding to different ways that Eve might attempt to break the system. The first of these is the *RSA factoring problem*: Given a number n that is known to be the product of two primes p and q, find p and q. Clearly if Eve can find p and q, then she can compute the decryption key d from the public encryption key e (in the same way that Alice did when generating the key) and subsequently decrypt all ciphertexts.

55.2 Computing $\phi(n)$

Eve doesn't really need to know the factors of n in order to compute d. It is enough for her to know $\phi(n)$. Computing $\phi(n)$ is no harder than factoring n since $\phi(n)$ is easily computed given the factors of n, but is it perhaps easier? If it were, then Eve would have a possible attack on RSA different from factoring n. It turns out that it is not easier, for if Even knows $\phi(n)$, then she can factor n. She simply sets up the system of quadratic equations

$$\begin{aligned} n &= pq \\ \phi(n) &= (p-1)(q-1) \end{aligned}$$

in two unknowns p and q and solves for p and q using standard methods of algebra.

55.3 Finding d

Another possibility is that Eve might somehow be able to compute d from e and n even without the ability to factor n or compute $\phi(n)$. That would represent yet another attack that couldn't be ruled out by the assumption that the RSA factoring problem is hard. However, that too is not possible, as we now show.

Suppose Eve knows n and e and is somehow able to obtain d. Then Eve can factor n by a probabilistic algorithm. The algorithm is presented in Figure1.

We begin by finding unique integers s and t such that $2^s t = ed - 1$ and t is odd. This is always possible since $ed - 1 \neq 0$. One way to find s and t is to express $ed - 1$ in binary. Then s is the number of trailing zeros and t is the value of the binary number that remains after the trailing zeros are removed. Since $ed - 1 \equiv 0 \pmod{\phi(n)}$ and $4 \mid \phi(n)$ (since both $p - 1$ and $q - 1$ are even), it follows that $s \geq 2$.

Now, for each a chosen at random from $\mathbf{Z}_n^*$, define a sequence $b_0, b_1, \ldots, b_s$, where $b_i = a^{2^i t} \bmod n$, $0 \leq i \leq s$. Each number in the sequence is the square of the number preceding it modulo n. The last number in the sequence is 1 by Euler's theorem and the fact that $\phi(n) \mid (ed-1)$. Since $1^2 \bmod n = 1$, every element of the sequence following the first 1 is also 1. Hence, the sequence consists of a (possibly empty) block of non-1 elements, following by a block of 1's.

It is easily verified that b_0 is the value of b when the innermost while loop is first entered, and b_k is the value of b after the k^{th} iteration of that loop. The loop executes at most $s - 1$ times since it terminates just before the first 1 is encounter, that is, when $b^2 \equiv 1 \pmod{n}$. At this time, $b = b_k$ is a square root of $1 \pmod{n}$.

To factor n:
  /* Find s, t such that $ed - 1 = 2^s t$ and t is odd */
  $s = 0;\ t = ed - 1;$
  while (t is even) $\{s{+}{+};\ t/{=}2;\}$
  /* Search for non-trivial square root of $1 \pmod n$ */
  do {
    /* Find a random square root b of $1 \pmod n$ */
    choose $a \in \mathbf{Z}_n^*$ at random;
    $b = a^t \bmod n$;
    while ($b^2 \not\equiv 1 \pmod n$) $b = b^2 \bmod n$;
  } while ($b \equiv \pm 1 \pmod n$);
  /* Factor n */
  $p = \gcd(b-1, n)$;
  $q = n/p$;
  return (p, q);
}

Figure 1: Algorithm for factoring n given d.

Over the reals, we know that each positive number has two square roots, one positive and one negative, and no negative numbers have real square roots. Over $\mathbf{Z}_n^*$ for $n = pq$, it turns out that $1/4$ of the numbers have square roots, and each number that has a square root actually has four. Since 1 does have a square root modulo n (itself), there are four possibilities for b: $\pm 1 \bmod n$ and $\pm r \bmod n$ for some $r \in \mathbf{Z}_n^*$, $r \not\equiv \pm 1 \pmod n$.

The do loop terminates if and only if $b \not\equiv \pm 1 \pmod n$. At that point we can factor n. Since $b^2 \equiv 1 \pmod n$, we have $n \mid (b^2 - 1) = (b+1)(b-1)$. But since $b \not\equiv \pm 1 \pmod n$, n does not divide $b + 1$ and n does not divide $b - 1$. Therefore, one of the factors of n divides $b + 1$ and the other divides $b - 1$. Hence, $\gcd(b-1, n)$ is a non-trivial factor of n.

It can be shown that there is at least a 0.5 probability that $b \not\equiv \pm 1 \pmod n$ for the b computed by this algorithm in response to a randomly chosen $a \in \mathbf{Z}_n^*$. Hence, the expected number of iterations of the do loop is at most 2. (See Evangelos Kranakis, *Primality and Cryptography*, Theorem 5.1 for details.)

Here's a simple example of the use of this algorithm. Suppose $n = 5 \times 11 = 55$, $e = 3$, and $d = 27$. (These are possible RSA values since $\phi(n) = 40$ and $ed = 81 \equiv 1 \pmod{40}$.) Then $ed - 1 = 80 = (1010000)_2$, so $s = 4$ and $t = 5$.

Now, suppose we take $a = 2$. We compute the sequence of b's as follows:

$$
\begin{aligned}
b_0 &= a^t \bmod n = 2^5 \bmod 55 = 32 \\
b_1 &= (b_0)^2 \bmod n = (32)^2 \bmod 55 = 1024 \bmod 55 = 34 \\
b_2 &= (b_1)^2 \bmod n = (34)^2 \bmod 55 = 1156 \bmod 55 = 1 \\
b_3 &= (b_2)^2 \bmod n = (1)^2 \bmod 55 = 1 \\
b_4 &= (b_3)^2 \bmod n = (1)^2 \bmod 55 = 1
\end{aligned}
$$

Since the last $b_i \neq 1$ in this sequence is 34, and $34 \not\equiv -1 \pmod{55}$, then 34 is a non-trivial square root of 1 modulo 55. It follows that $\gcd(34 - 1, 55) = 11$ is a prime divisor of n.

55.4 Finding plaintext

Eve isn't really interested in factoring n, computing $\phi(n)$, or finding d, except as a means to read Alice's secret messages. Hence, the problem we would like to be hard is the problem of computing the plaintext m given an RSA public key (n, e) and a ciphertext c. The above shows that this problem is no harder than factoring n, computing $\phi(n)$, or finding d, but it does not rule out the possibility of some clever way of decrypting messages without actually finding the decryption key. Perhaps there is some feasible probabilistic algorithm that finds m with non-negligible probability, maybe not even for all ciphertexts c but for some non-negligible fraction of them. Such a method would "break" RSA and render it useless in practice. No such algorithm has been found, but neither has the possibility been ruled out, even under the assumption that the factoring problem itself is hard.