

CPSC 467b: Cryptography and Computer Security

Lecture 11

Michael J. Fischer

Department of Computer Science
Yale University

February 15, 2010

- 1 Primality Tests (cont.)
 - Weak tests of compositeness
 - Reformulation of weak tests of compositeness
 - Examples of weak tests
- 2 Chinese Remainder Theorem
 - Homomorphic property of χ
 - RSA Decryption Works for All of Z_n
- 3 RSA Security
 - Factoring n
 - Computing $\phi(n)$
 - Finding d directly
 - Finding plaintext

Weak tests

Recall: A *weak test of compositeness* $T(n, a)$ is only required to have many witnesses to the correct answer when n is composite.

When n is prime, a weak test always answers '?', so there are no witnesses to n being prime.

Hence, the test either outputs '**composite**' or '?' but never '**prime**'.

An answer of '**composite**' means that n is definitely **composite**, but these tests can never say for sure that n is prime.

Use of a weak test of compositeness

Let $T(n, a)$ be a weak test of compositeness. Algorithm P_2 is a “best effort” attempt to prove that n is composite.

Since T is a weak test, we can slightly simplify P_2 .

Algorithm $P_2(n, t)$:

```
repeat  $t$  times {  
    Generate a random helper string  $a$ ;  
    if ( $T(n, a) = \text{'composite'}$ ) return  $\text{'composite'}$ ;  
}  
return  $\text{'?'}$ ;
```

P_2 returns **composite** just in case it succeeds in finding a helper string a for which the test succeeds.

Such a string a is a *witness* to the compositeness of n .

Finding a random prime

P_2 is used in generating a random prime.

Algorithm GenPrime(k):

```
const int t=20;  
do {  
    Generate a random  $k$ -bit integer  $x$ ;  
} while (  $P_2(x, t) == \text{'composite'}$  );  
return  $x$ ;
```

The number x that GenPrime() returns has the property that P_2 failed to find a witness to its compositeness after t trials, but there is still the possibility that x is composite.

Boolean test of compositeness

A Boolean function $\tau(n, a)$ can be interpreted as a weak test of compositeness by taking true to mean '**composite**' and false to mean '?'.

We may write $\tau_a(n)$ to mean $\tau(n, a)$.

- If $\tau_a(n) = \text{true}$, we say that τ_a *succeeds on n* , and a is a *witness to the compositeness of n* .
- If $\tau_a(n) = \text{false}$, then τ_a *fails* and gives no information about the compositeness of n .

Clearly, if n is prime, then τ_a fails on n for all a , but if n is composite, then τ_a may succeed for some values of a and fail for others.

Useful tests

A test of compositeness τ is *useful* if

- there is a feasible algorithm that computes $\tau(n, a)$;
- for every composite number n , $\tau_a(n)$ succeeds for a fraction $c > 0$ of the help strings a .

Suppose for simplicity that $c = 1/2$ and one computes $\tau_a(n)$ for 100 randomly-chosen values for a .

- If any of the τ_a succeeds, we have a proof a that n is composite.
- If all fail, we don't know whether or not n is prime or composite. But we do know that if n is composite, the probability that all 100 tests will fail is only $1/2^{100}$.

Application to RSA

In practice, we use $\text{GenPrime}(k)$ to choose RSA primes p and q , where the constant t is set according to the number of witnesses and the confidence levels we would like to achieve.

For $c = 1/2$, using $t = 20$ trials gives us a failure probability of about one in a million (for each of p and q), or about two in a million of generating a bad RSA modulus. t can be increased if this risk of failure is deemed to be too large.

Finding weak tests of compositeness

We still need to find useful weak tests of compositeness.

We begin with two simple examples. While neither is useful, they illustrate some of the ideas behind the useful tests that we will present later.

The division test $\delta_a(n)$

Let

$$\delta_a(n) = (2 \leq a \leq n - 1 \text{ and } a|n).$$

Test δ_a succeeds on n iff 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 useful since the fraction of witnesses to n 's compositeness is exponentially small.

For example, if $n = pq$ for p, q prime, then the *only* witnesses are p and q , and the only tests that succeed are δ_p and δ_q .

The Fermat test $\zeta_a(n)$

Let

$$\zeta_a(n) = (2 \leq a \leq n - 1 \text{ and } a^{n-1} \not\equiv 1 \pmod{n}).$$

By Fermat's theorem, if n is prime and $\gcd(a, n) = 1$, then $a^{n-1} \equiv 1 \pmod{n}$.

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 useful, we would need to know that every composite number n has a constant fraction of witnesses.

Carmichael numbers (Fermat pseudoprimes)

Unfortunately, there are certain composite numbers n called *Carmichael numbers*¹ for which there are no witnesses, and all of the tests ζ_a fail. Such n are fairly rare, but they do exist. The smallest such n is $561 = 3 \cdot 11 \cdot 17$.²

Hence, Fermat tests are not useful tests of compositeness according to our definition, and they are unable to distinguish Carmichael numbers from primes.

We defer discussion of weak tests that are both valid and useful until we have developed some more needed number theory.

¹Carmichael numbers are sometimes called Fermat pseudoprimes.

²See http://en.wikipedia.org/wiki/Carmichael_number for further information.

Systems of congruence equations

Theorem (Chinese remainder theorem)

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

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

(1) has a unique solution $x \in \mathbf{Z}_n$.

^aThis means that $\gcd(n_i, n_j) = 1$ for all $1 \leq i < j \leq k$.

How to solve congruence equations

To solve for x , let

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

and compute $M_i = N_i^{-1} \pmod{n_i}$, for $1 \leq i \leq k$.

$N_i^{-1} \pmod{n_i}$ exists since $\gcd(N_i, n_i) = 1$. (Why?)

We can compute N_i^{-1} by solving the associated Diophantine equation as described in Lecture 10.

The solution to (1) is

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

Correctness

Lemma

$$M_j N_j \equiv \begin{cases} 1 \pmod{n_i} & \text{if } j = i; \\ 0 \pmod{n_i} & \text{if } j \neq i. \end{cases}$$

Proof.

$M_i N_i \equiv 1 \pmod{n_i}$ since $M_i = N_i^{-1} \pmod{n_i}$.
If $j \neq i$, then $M_j N_j \equiv 0 \pmod{n_i}$ since $n_i | N_j$. □

It follows from the lemma and the fact that $n_i | n$ that

$$x \equiv \sum_{i=1}^k a_i M_i N_i \equiv a_i \pmod{n_i} \quad (3)$$

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

Uniqueness

To see that the solution is unique in \mathbf{Z}_n , let

$\chi : \mathbf{Z}_n \rightarrow \mathbf{Z}_{n_1} \times \dots \times \mathbf{Z}_{n_k}$ be the mapping

$$x \mapsto (x \bmod n_1, \dots, x \bmod n_k).$$

χ is a surjection³ since $\chi(x) = (a_1, \dots, a_k)$ iff x satisfies (1).

Since also $|\mathbf{Z}_n| = |\mathbf{Z}_{n_1} \times \dots \times \mathbf{Z}_{n_k}|$, χ is a bijection, and there is only one solution to (1) in \mathbf{Z}_n .

³A *surjection* is an onto function.

An alternative proof of uniqueness

A less slick but more direct way of seeing uniqueness is to suppose that $x = u$ and $x = v$ are both solutions to (1).

Then $u \equiv v \pmod{n_i}$, so $n_i | (u - v)$ for all i .

By the pairwise relatively prime condition on the n_i , it follows that $n | (u - v)$, so $u \equiv v \pmod{n}$. Hence, the solution is unique in \mathbf{Z}_n .

Operations on tuples

The bijection χ establishes a one-to-one correspondence between members of \mathbf{Z}_n and k -tuples (a_1, \dots, a_k) in $\mathbf{Z}_{n_1} \times \dots \times \mathbf{Z}_{n_k}$.
 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 corresponding operation on each of the components of $\chi(x)$.

Let $\odot \in \{+, -, \times\}$, $\chi(x) = (a_1, \dots, a_k)$, $\chi(y) = (b_1, \dots, b_k)$.
 Then

$$\begin{aligned} \chi((x \odot y) \bmod n) \\ = ((a_1 \odot b_1) \bmod n_1, \dots, (a_k \odot b_k) \bmod n_k). \end{aligned} \quad (4)$$

This follows because $n_i \mid n$, so

$$((x \odot y) \bmod n) \bmod n_i = (a_i \odot b_i) \bmod n_i.$$

Example

Let $n_1 = 4$, $n_2 = 3$, $n_3 = 7$, so $n = 84$.

$$\chi(15) = (3, 0, 1)$$

$$\chi(23) = (3, 2, 2)$$

$$\begin{aligned}\chi((15 \times 23) \bmod n) &= \chi(345 \bmod 84) = \chi(9) \\ &= (9 \bmod 4, 9 \bmod 3, 9 \bmod 7) = (1, 0, 2)\end{aligned}$$

. Check:

$$((3 \times 3) \bmod 4 = (9 \bmod 4) = 1$$

$$((0 \times 2) \bmod 3 = (0 \bmod 3) = 0$$

$$((1 \times 2) \bmod 7 = (2 \bmod 7) = 2.$$

An application of the Chinese Remainder Theorem

We showed previously 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 use the Chinese Remainder Theorem to supply this missing piece.

Theorem (RSA encryption is invertible over all of \mathbf{Z}_n)

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

Proof

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 will show that

$$\begin{aligned}a^{ed} &\equiv a \pmod{p} \\ b^{ed} &\equiv b \pmod{q}\end{aligned}\tag{7}$$

Combining (6) with (7) yields

$$\begin{aligned}m^{ed} &\equiv a \pmod{p} \\ m^{ed} &\equiv b \pmod{q}\end{aligned}\tag{8}$$

Proof (cont.)

From (5) and (8), we see that both m and m^{ed} are solutions to the system of equations

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

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

It remains to show

$$\begin{aligned}a^{ed} &\equiv a \pmod{p} \\b^{ed} &\equiv b \pmod{q}\end{aligned}\tag{7}$$

Proof (cont.)

We first 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,

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

Similarly, $b^{ed} \equiv b \pmod{q}$. □

Attacks on RSA

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.

- Factoring n
- Computing $\phi(n)$
- Finding d directly
- Finding plaintext

We examine each in turn and look at their relative computational difficulty.

RSA factoring problem

Definition (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, n) (in the same way that Alice did when generating the key).

This completely breaks the system, for now Eve has the same power as Bob to decrypt all ciphertexts.

This problem is a special case of the general factoring problem. It is believed to be intractable, although it is not known to be NP-complete.

$\phi(n)$ problem

Definition ($\phi(n)$ problem)

Given a number n that is known to be the product of two primes p and q , compute $\phi(n)$.

Eve doesn't really need to know the factors of n in order to break RSA. It is enough for her to know $\phi(n)$, since that allows her to compute $d = e^{-1} \pmod{\phi(n)}$.

Computing $\phi(n)$ is no easier than factoring n . Given n and $\phi(n)$, Eve can factor n by solving the system of quadratic equations

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

for p and q using standard methods of algebra.

Decryption exponent problem

Definition (Decryption exponent problem)

Given an RSA public key (e, n) , find the decryption exponent d .

Eve might somehow be able to find d directly from e and n even without the ability to factor n or to 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.

Factoring n knowing e and d

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 $ed - 1 \neq 0$.

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 $s \geq 2$.

Square roots of 1 (mod n)

Over the reals, 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 $n = pq$, $1/4$ 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 :

$$\pm 1 \pmod{n} \quad \text{and} \quad \pm r \pmod{n}$$

for some $r \in \mathbf{Z}_n^*$, $r \not\equiv \pm 1 \pmod{n}$.

Finding a square root of 1 (mod n)

Using randomization to find a square root of 1 (mod n).

- Choose random $a \in \mathbf{Z}_n^*$.
- Define a sequence b_0, b_1, \dots, 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 (mod n).
- The last number in the sequence is $b_s = a^{ed-1} \bmod n$.
- Since $ed \equiv 1 \pmod{\phi(n)}$, it follows using Euler's theorem that $b_s \equiv 1 \pmod{n}$.
- 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 one or more 1's.

Using a non-trivial square root of unity to factor n

Suppose $b^2 \equiv 1 \pmod{n}$. Then $n \mid (b^2 - 1) = (b + 1)(b - 1)$.

Suppose further that $b \not\equiv \pm 1 \pmod{n}$. Then $n \nmid (b + 1)$ and $n \nmid (b - 1)$.

Therefore, one of the factors of n divides $b + 1$ and the other divides $b - 1$.

Hence, $p = \gcd(b - 1, n)$ is a non-trivial factor of n .

The other factor is $q = n/p$.

Randomized factoring algorithm knowing d

```

Factor  $(n, e, d)$  { //finds  $s, t$  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 (mod  $n$ )
  do {
    // Find a random square root  $b$  of 1 (mod  $n$ )
    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  $(p, q)$ ;
}

```


Notes on the algorithm

Notes:

- 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.
- The inner loop executes at most $s - 1$ times since it terminates just before the first 1 is encountered, that is, when $b^2 \equiv 1 \pmod{n}$.
- At that time, $b = b_k$ is a square root of 1 \pmod{n} .
- The outer do loop terminates if and only if $b \not\equiv \pm 1 \pmod{n}$. At that point we can factor n .

The probability that $b \not\equiv \pm 1 \pmod{n}$ for a randomly chosen $a \in \mathbf{Z}_n^*$ is at least 0.5 .⁴ Hence, the expected number of iterations of the do loop is at most 2.

⁴(See Evangelos Kranakis, *Primality and Cryptography*, Theorem 5.1 for details.)

Example

Suppose $n = 55$, $e = 3$, and $d = 27$.⁵

Then $ed - 1 = 80 = (1010000)_2$, so $s = 4$ and $t = 5$.

Now, suppose we choose $a = 2$. We compute the sequence of b 's.

$$b_0 = a^t \bmod n = 2^5 \bmod 55 = 32$$

$$b_1 = (b_0)^2 \bmod n = (32)^2 \bmod 55 = 1024 \bmod 55 = 34$$

$$b_2 = (b_1)^2 \bmod n = (34)^2 \bmod 55 = 1156 \bmod 55 = 1$$

$$b_3 = (b_2)^2 \bmod n = (1)^2 \bmod 55 = 1$$

$$b_4 = (b_3)^2 \bmod n = (1)^2 \bmod 55 = 1$$

The last $b_i \neq 1$ in this sequence is $b_1 = 34 \not\equiv -1 \pmod{55}$, so 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 .

⁵These are possible RSA values since $n = 5 \times 11$, $\phi(n) = 4 \times 10 = 40$, and $ed = 81 \equiv 1 \pmod{40}$.

Known ciphertext attack against RSA

Eve isn't really interested in factoring n , computing $\phi(n)$, or finding d , except as a means to read Alice's secret messages.

The problem we would like to be hard is

Definition (Known ciphertext problem)

Given an RSA public key (n, e) and a ciphertext c , **find the plaintext message m .**

Hardness of known ciphertext attack

A known ciphertext attack on RSA 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.**