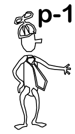
15-251

Great Theoretical Ideas in Computer Science

Raising numbers to powers, Cyrptography and RSA,

Lecture 14 (October 7, 2010)



$^{p-1} \equiv_p 1$

How do you compute…

5^8 using few multiplications?

First idea:

$5 \quad 5^2 \quad 5^3 \quad 5^4 \quad 5^5 \quad 5^6 \quad 5^7 \quad 5^8$

$= 5*5 \quad 5^2*5$

How do you compute…

5^8

Better idea:

$5 \quad 5^2 \quad 5^4 \quad 5^8$

$= 5*5 \quad 5^2*5^2 \quad 5^4*5^4$

Used only 3 mults instead of 7 !!!

Repeated squaring calculates a^{2^k} in k multiply operations

compare with
$(2^k – 1)$ multiply
operations
used by the naïve method

How do you compute…

5^{13}

Use repeated squaring again?

$5 \quad 5^2 \quad 5^4 \quad 5^8 \quad$ ~~5^{16}~~

too high! what now?

assume no divisions allowed…

How do you compute…

5^{13}

Use repeated squaring again?

5 5^2 5^4 5^8

Note that 13 = 8+4+1 ($13_{10} = (1101)_2$)

So $a^{13} = a^8 * a^4 * a^1$

Two more multiplies!

To compute a^m

Suppose $2^k \le m < 2^{k+1}$

a a^2 a^4 a^8 . . . a^{2^k}

This takes k multiplies

Now write m as a sum of distinct powers of 2

say, $m = 2^k + 2^{i_1} + 2^{i_2} \ldots + 2^{i_t}$

$a^m = a^{2^k} * a^{2^{i1}} * \ldots * a^{2^{it}}$

at most k more multiplies

Hence, we can compute a^m while performing at most $2 \lfloor \log_2 m \rfloor$ multiplies

How do you compute…

5^{13} (mod 11)

First idea: Compute 5^{13} using 5 multiplies

5 5^2 5^4 5^8 5^{12} 5^{13} = 1 220 703 125

= 5^8*5$5^{12}$*5

then take the answer mod 11

1220703125 (mod 11) = 4

How do you compute…

5^{13} (mod 11)

Better idea: keep reducing the answer mod 11

5	5^2	5^4	5^8	5^{12}	5^{13}
	25		´$_{11}$ 81	´$_{11}$ 36	´$_{11}$ 15
	´$_{11}$ 3	´$_{11}$ 9	´$_{11}$ 4	´$_{11}$ 3	´$_{11}$ 4

Hence, we can compute a^m (mod n) while performing at most $2 \lfloor \log_2 m \rfloor$ multiplies

where each time we multiply together numbers with $\lfloor \log_2 n \rfloor + 1$ bits

How do we implement this?

Let's use my favorite programming language – Ocaml.

It's a functional language that automatically infers the types of variables. It compiles to fast code. It has an interactive shell so that you can play with the functions you've written. (Similar to SML which you will learn about in 15-212 or 15-150.)

```
(* compute a to the pth power modulo n *)

let rec powermod a p n =
  let sq x = (x*x) mod n in
    if p=0 then 1 else
      let x = sq (powermod a (p/2) n) in
        if p mod 2 = 0 then x else (a*x) mod n
```

How do you compute…

$5^{121242653}$ (mod 11)

The current best idea would still need about 54 calculations

answer = 4

Can we exponentiate any faster?

OK, need a little more number theory for this one…

First, recall…

$Z_n = \{0, 1, 2, \ldots, n-1\}$

$Z_n^* = \{x \in Z_n \mid GCD(x,n) = 1\}$

Fundamental lemmas mod n:

If $(x \equiv_n y)$ and $(a \equiv_n b)$. Then

1) $x + a \equiv_n y + b$
2) $x * a \equiv_n y * b$
3) $x - a \equiv_n y - b$
4) $cx \equiv_n cy \Rightarrow a \equiv_n b$ [i.e., if c in Z_n^*]

Euler Phi Function $\Phi(n)$

$\Phi(n)$ = size of Z_n^*

p prime $\Rightarrow \Phi(p) = p-1$

p, q distinct primes $\Rightarrow$
$\Phi(pq) = (p-1)(q-1)$

~~Fundamental lemma of powers?~~

If $(x \equiv_n y)$
Then $a^x \equiv_n a^y$?

NO!

$(2 \equiv_3 5)$, but it is not
the case that: $2^2 \equiv_3 2^5$

(Correct) Fundamental lemma of powers.

If $a \in Z_n^*$ and $x \equiv_{\Phi(n)} y$ then $a^x \equiv_n a^y$

Equivalently,

for $a \in Z_n^*$, $a^x \equiv_n a^{x \bmod \Phi(n)}$

How do you compute…

$5^{121242653}$ (mod 11)

121242653 (mod 10) = 3

5^3 (mod 11) = 125 mod 11 = 4

Why did we take mod 10?

for $a \in Z_n^*$, $a^x \equiv_n a^{x \bmod \Phi(n)}$

Hence, we can compute
a^m (mod n)
while performing at most
$2 \lfloor \log_2 \Phi(n) \rfloor$ multiplies

where each time we multiply
together numbers
with $\lfloor \log_2 n \rfloor + 1$ bits

$343281^{327847324}$ mod 39

Step 1: reduce the base mod 39

Step 2: reduce the exponent mod $\Phi(39) = 24$

NB: you should check that gcd(343280,39)=1 to use lemma of powers

Step 3: use repeated squaring to compute 3^4, taking mods at each step

(Correct) Fundamental lemma of powers.

If $a \in Z_n^*$ and $x \equiv_{\Phi(n)} y$ then $a^x \equiv_n a^y$

Equivalently,

for $a \in Z_n^*$, $a^x \equiv_n a^{x \bmod \Phi(n)}$

How do you prove the lemma for powers?

Use Euler's Theorem

For $a \in Z_n^*$, $a^{\Phi(n)} \equiv_n 1$

Corollary: Fermat's Little Theorem

For p prime, $a \in Z_p^* \Rightarrow a^{p-1} \equiv_p 1$

Proof of Euler's Theorem: for $a \in Z_n^*$, $a^{\Phi(n)} \equiv_n 1$

Define $a Z_n^* = \{a *_n x \mid x \in Z_n^*\}$ for $a \in Z_n^*$

By the cancellation property, $Z_n^* = aZ_n^*$

$\prod x \equiv_n \prod ax$ [as x ranges over Z_n^*]

$\prod x \equiv_n \prod x \; (a^{\text{size of } Zn^*})$ [Commutativity]

$1 =_n a^{\text{size of } Zn^*}$ [Cancellation]

$a^{\Phi(n)} =_n 1$

Please remember

Euler's Theorem

For $a \in Z_n^*$, $a^{\Phi(n)} \equiv_n 1$

Corollary: Fermat's Little Theorem

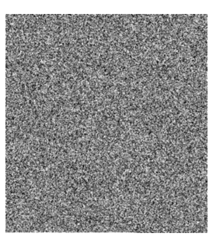
For p prime, $a \in Z_p^* \Rightarrow a^{p-1} \equiv_p 1$

Basic Cryptography

One Time Pads

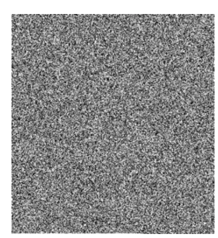
The meeting
will be
in the town hall
at
midnight!

Come with your
parole officer!



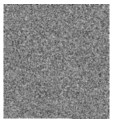
One Time Pads

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam posuere dolor et neque. Vivamus sed lorem. Vivamus gravida quam at arcu. Sed auctor velit at dui. Donec venenatis augue ut nibh. Aliquam auctor. Aliquam volutpat. In ligula velit, scelerisque ac, lacinia quis, facilisis vitae, diam. Pellentesque porttitor nunc eget lectus. Nullam velit lectus, iaculis laoreet, porta sed, pellentesque in, velit. Sed eu elit. Proin orci. Donec dignissim tempor erat. Quisque gravida.

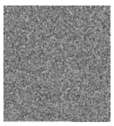


they give perfect security!

But reuse is bad



XOR



=

Can do other attacks as well

Agreeing on a secret

One time pads rely on having a shared secret!

Alice and Bob have never talked before
but they want to agree on a secret…

How can they do this?

A couple of small things

A value g in Z_n^* "generates" Z_n^* if
$g, g^2, g^3, g^4, \ldots, g^{\Phi(n)}$
contains all elements of Z_n^*

Diffie-Hellman Key Exchange

Alice:
Picks prime p, and a generator g in Z_p^*
Picks random a in Z_p^*
Sends over p, g, g^a (mod p)

Bob:
Picks random b in Z_p^*, and sends over g^b (mod p)

Now both can compute g^{ab} (mod p)

What about Eve?

Alice:
Picks prime p, and a value g in Z_p^*
Picks random a in Z_p^*
Sends over p, g, g^a (mod p)

Bob:
Picks random b in Z_p^*, and sends over g^b (mod p)

Now both can compute g^{ab} (mod p)

If Eve's just listening in,
she sees p, g, g^a, g^b

It's believed that computing g^{ab} (mod p) from just
this information is not easy…

also, discrete logarithms seem hard

Discrete-Log:
Given p, g, g^a (mod p), compute a

How fast can you do this?

If you can do discrete-logs fast,
you can solve the Diffie-Hellman problem fast.

How about the other way? If you can break the DH
key exchange protocol, do discrete logs fast?

Diffie Hellman requires both parties
to exchange information to share a secret

can we get rid of this assumption?

The RSA Cryptosystem

Our dramatis personae

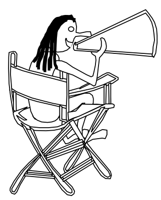

Rivest


Shamir


Adleman


Euler

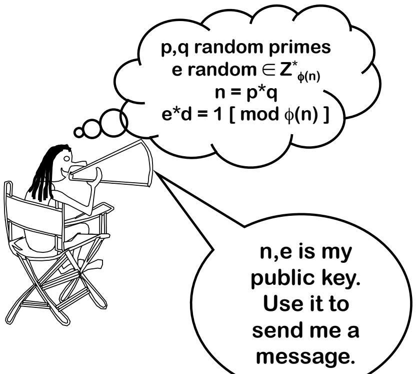

Fermat

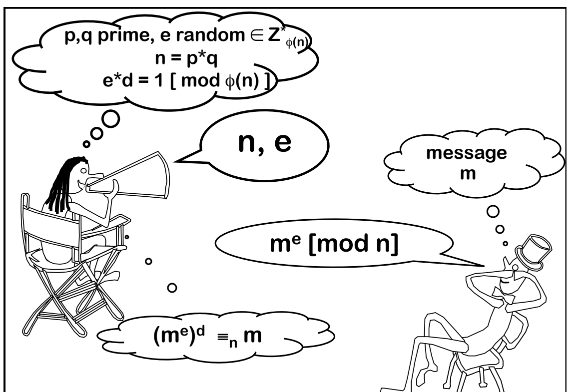


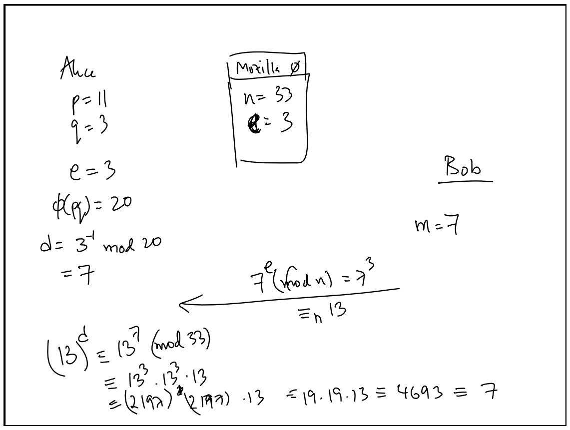
Pick secret, random large primes: p,q
Multiply n = p*q
“Publish”: n

$\phi(n) = \phi(p)\,\phi(q) = (p-1)*(q-1)$
Pick random $e \in Z^*_{\phi(n)}$
“Publish”: e

Compute d = inverse of e in $Z^*_{\phi(n)}$
Hence, e*d = 1 [mod $\phi(n)$]
“Private Key”: d







How hard is cracking RSA?

If we can factor products of two large primes, can we crack RSA?

If we know n and Φ(n), can we crack RSA?

How about the other way? Does cracking RSA mean we must do one of these two?

We don't know (yet)…

How do we generate large primes?

The density of primes is about 1/ln(n). So that if we can efficiently test the primality of a number, then we can generate primes fast.

Answer: The Miller-Rabin primality test.

(Gary Miller is one of our professors.)

Miller-Rabin test

The idea is to use a "converse" of Fermat's Theorem. We know that:

$$a^{n-1} \equiv_n 1$$

for any prime n and any a in [2, n-1]. What if we try this for some number a and it fails. Then we know that n is NOT prime. Miller-Rabin is based on this idea.

Say we write n-1 as $d * 2^s$ where d is odd.

Consider the following sequence of numbers mod n:

$$a^d, a^{2d}, a^{4d} \ldots a^{d*2^{(s-1)}}, a^{d*2^s} = a^{n-1} \equiv_n 1$$

Each element is the square of the previous one.

$$a^d, a^{2d}, a^{4d} \ldots a^{d*2^{(s-1)}}, a^{d*2^s} = a^{n-1} \equiv_n 1$$

If n is prime, then at some point the sequence hits 1 and stays there from then on.

The interesting point is: what is the number right before the first 1. If n is prime this MUST BE n-1.

Miller-Rabin Test

To test a number n, we pick a random a and generate the above sequence. If the sequence does not hit 1, then n is composite. If there's an element before the first 1 and it's not n-1, then n is composite.

Otherwise n is "probably prime".

Miller-Rabin Analysis

If n is composite, then with a random a, the Miller-Rabin algorithm says "composite" with probability at least 3/4 .

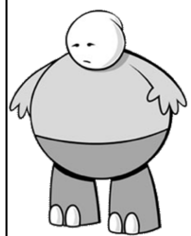
So if we run the test 30 times and it never says "composite" then n is prime with "probability" $1-2^{-60}$

In other words it's more likely that you'll win the lottery three days in a row than that this is giving a wrong answer.

i.e. not bloody likely.

This ocaml implementation of the Miller-Rabin test does not pick random random witnesses, but rather uses 2, 3, 5, and 7. It's guaranteed to work up to about 2 billion. See the accompanying file big_number.ml for a full high precision implementation of Miller-Rabin with random witnesses.

```
let miller_rabin n =
  if n<=10 then (n=2 or n=3 or n=5 or n=7) else
    if (n mod 2=0 or n mod 3=0 or n mod 5=0 or n mod 7=0) then false else
      let rec remove_twos m =
        let h = m/2 in
          if (h+h < m) then (0,m) else
            let (s,d) = remove_twos h in (s+1,d)
      in
      let (s,d) = remove_twos (n-1) in   (* so d*2^s = n-1 *)
      let is_witness_to_compositeness a =
        let x = powermod a d n in
          if x=1 or x=(n-1) then false else
            let rec loop x r =               (* at this point x = a^(d * 2^r) mod n *)
              if x=1 or r=s then true else
                if x = (n-1) then false else
                  loop ((x*x) mod n) (r+1)
            in loop ((x*x) mod n) 1
      in
        if (is_witness_to_compositeness 2) then false
        else if (is_witness_to_compositeness 3) then false
        else if (is_witness_to_compositeness 5) then false
        else if (is_witness_to_compositeness 7) then false
        else true
```



Here's What You Need to Know…

Fast exponentiation

Fundamental lemma of powers
- **Euler phi function $\phi(n) = |Z_n^*|$**
- **Euler's theorem**
- **Fermat's little theorem**

Diffie-Hellman Key Exchange

RSA algorithm

Generating Large Primes