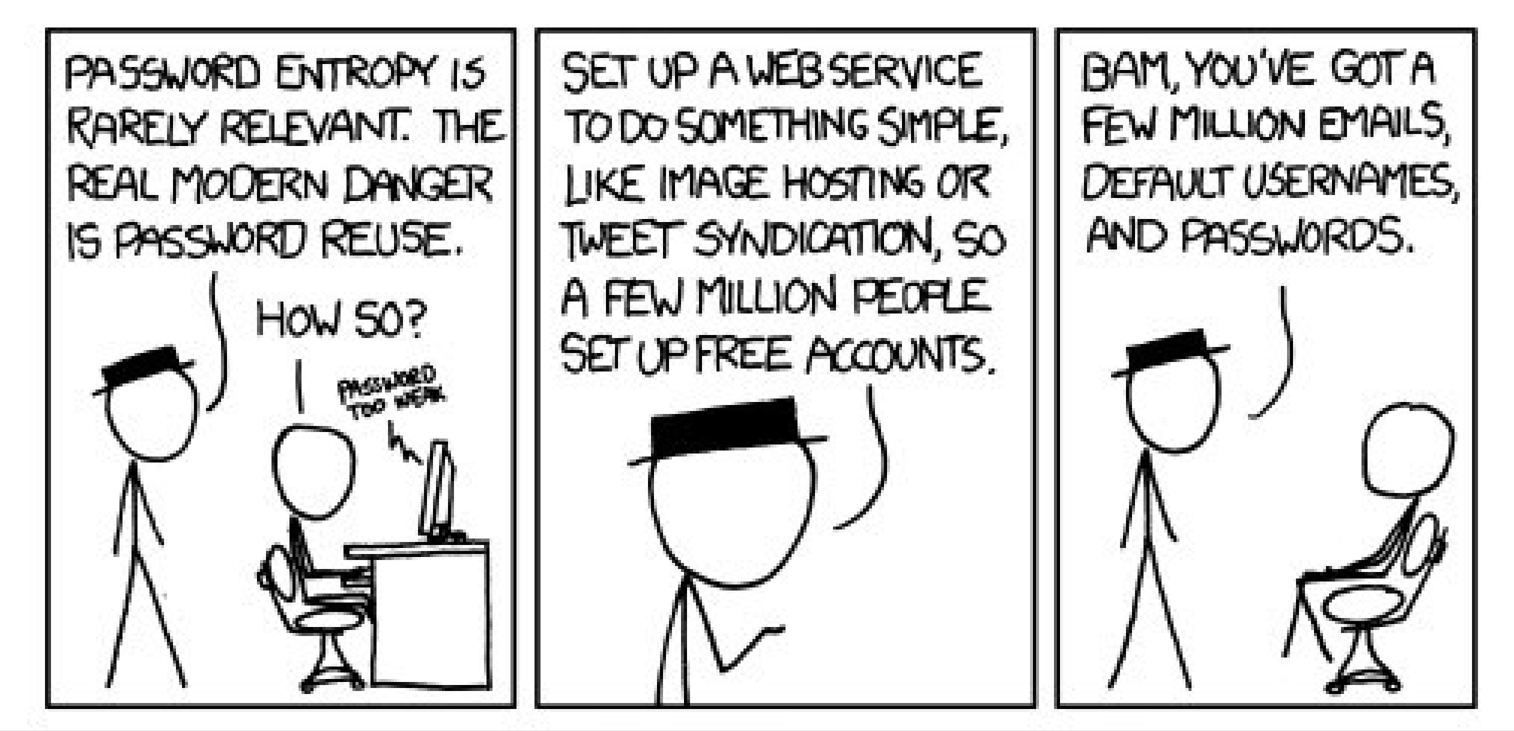
Prime numbers (cryptography)



Announcements

Test next Tuesday

Homework due Sunday

GCD

Let d | a mean $\exists k \in \mathbb{Z} \text{ such that } a = k \cdot d$

Example:
5 | 10, as 10 = 2 * 5

The greatest common divisor between a and b is:
gcd(a,b) = max x s.t. x | a and x | b

GCD

Oddly, another definition of gcd is:

$$\min_c c = a \cdot x + b \cdot y$$
$$\text{such that } c > 0 \text{ and } x, y \in \mathbb{Z}$$

gcd also has properties:

1. gcd(an, bn) = n gcd(a,b)
2. if n | ab and gcd(a,n) = 1, then n | b
3. if gcd(a,p)=1 and gcd(b,p)=1,
   then gcd(ab,p) = 1

GCD

We can recursively find gcd by:

gcd(a, b)
if b == 0, return a;
else, return gcd(b, a mod b)

a mod b will always decrease,
thus this will terminate

Modular linear equations

Suppose we wanted to solve:

$a x \bmod n = b$

E.g. $18 x \bmod 80 = 33$

How would you do this?

Modular linear equations

Let d = gcd(a, n)

Let x' and y' be integer solutions to:

    d = a*x' + n*y'

If d | b, then:

  There are d solutions, namely:

    for i = 0 to d-1

      print x'(b/d) + i(n/d) mod n

else, no solutions

Chinese remainder theorem

Let $n = n_1 * n_2 * \ldots * n_k$, where n_i is pairwise relatively prime

Then there is a unique solution for x:

$x \bmod n_i = a_i$

for all $i=1, 2, \ldots k$, when $x < n$

Chinese remainder theorem

This is a specific extension of solving a single equation (mod n)

The “loopy” nature of modulus comes in handy many places

Some implementations of FFT use the Chinese remainder theorem

Chinese remainder theorem

You can compute this solution as:

mod n_i for finding m_i^{-1}
not a math op

Let $m_i = n/n_i$

Then $c_i = m_i(m_i^{-1} \bmod n_i)$

Then $x = \sum c_i * a_i \bmod n$

(m_i^{-1} is such that $m_i * m_{i-1} \bmod n_i = 1$)

Chinese remainder theorem

Example, solve for x:

$x \bmod 5 = 2\ (a_1)$

$x \bmod 11 = 7\ (a_2)$

$n = 55,\ m_1 = 11,\ m_2 = 5$

$m_1^{-1} = 1,\ m_2^{-1} = 9$

$c_1 = 11*1 = 11,\ c_2 = 5*9 = 45$

$x = 11*2 + 7*45 \bmod 55 = 337\%55 = \underline{7}$

CRT vs. interpolation

There is actually some similarity between the CRT and interpolation

Both of them find a partial answer that simply modifies one sub-problem

Then combines these partial answers

CRT vs. interpolation

Find polynomial given 3 points:
(0,1), (1, 4), (2, 4)

(x-0)(x-1) is zero on x=0,1 (first 2)
2(x-0)(x-1) is correct for last (x=2)

Combine by adding up a polynomial for each point (not effecting others)

CRT vs. interpolation

Solve k systems of linear modular equations
$x \bmod n_1 = a_1$, $x \bmod n_2 = a_2$, ... $x \bmod n_k = a_k$

If $n = n_1 * n_2 * ... * n_k$, and $m_i = n/n_i$, then m_i has no effect on $x \bmod n_j$ for any j except i (as $n_j \mid m_i$)

So we find c_i such that $c_i m_i = x \pmod{n_i}$

Then add these terms together (not effect other)

RSA Encryption

RSA person A has two keys:

P_A = public key

S_A = secret key (private key)

The key is that these functions are inverse, namely for some message M:

$$P_A(S_A(M)) = S_A(P_A(M)) = M$$

RSA Encryption

Thus, if person B wants to send a secret message to person A, they do:

1. Encrypt the message using public key: $C = P_A(M)$
2. Then A can decrypt it using the secret key: $M = S_A(C)$

RSA Encryption

If A does not share S_A, no one else knows the proper way to decrypt C

$P_A(P_A(M)) \neq M$

... and ...

S_A not <u>easily computable</u> from P_A

RSA Encryption

RSA algorithm:

1. Select two large primes p, q ($p \neq q$)
2. Let $n = p * q$
3. Let e be: $gcd(e, (p - 1)*(q - 1)) = 1$
4. Let d be: $e*d \bmod (p-1)*(q-1) = 1$
(use “extended euclidean” in book)
5. Public key: $P = (e, n)$
6. Secret key: $S = (d, n)$

RSA Encryption

Specifically:

$P_A(M) = M^e \bmod n$

$S_A(C) = C^d \bmod n$

A key assumption is that $M < n$, as we want: $M \bmod n = M$

Pick large p,q or encode per byte

RSA Encryption

Example: p=7, q=11... n = p*q = 77

e=13 (does not need to be prime) as gcd(13,(7-1)(11-1))=gcd(13,60) = 1

d=37 as 13*37 mod 60 = 1

If M = 20, then...

C = 20^{13} mod 77 = 69

C = 69, 69^{37} mod 77 = 20

RSA Encryption + CRT

Computing large powers can require a lot of processor power

Can more efficiently get the result with Chinese remainder theorem: (backwards)
Have: number mod product
Want: smaller system of equations

RSA Encryption + CRT

Using CRT:

$m1 = C^{d \bmod p-1} \bmod p$ // less compute

$m2 = C^{d \bmod q-1} \bmod q$ // much smaller

$qI = q^{-1} \bmod p$

$h = qI * (m1 - m2)$

$m = m2 + h*q$

(see: rsa.cpp)

Primes

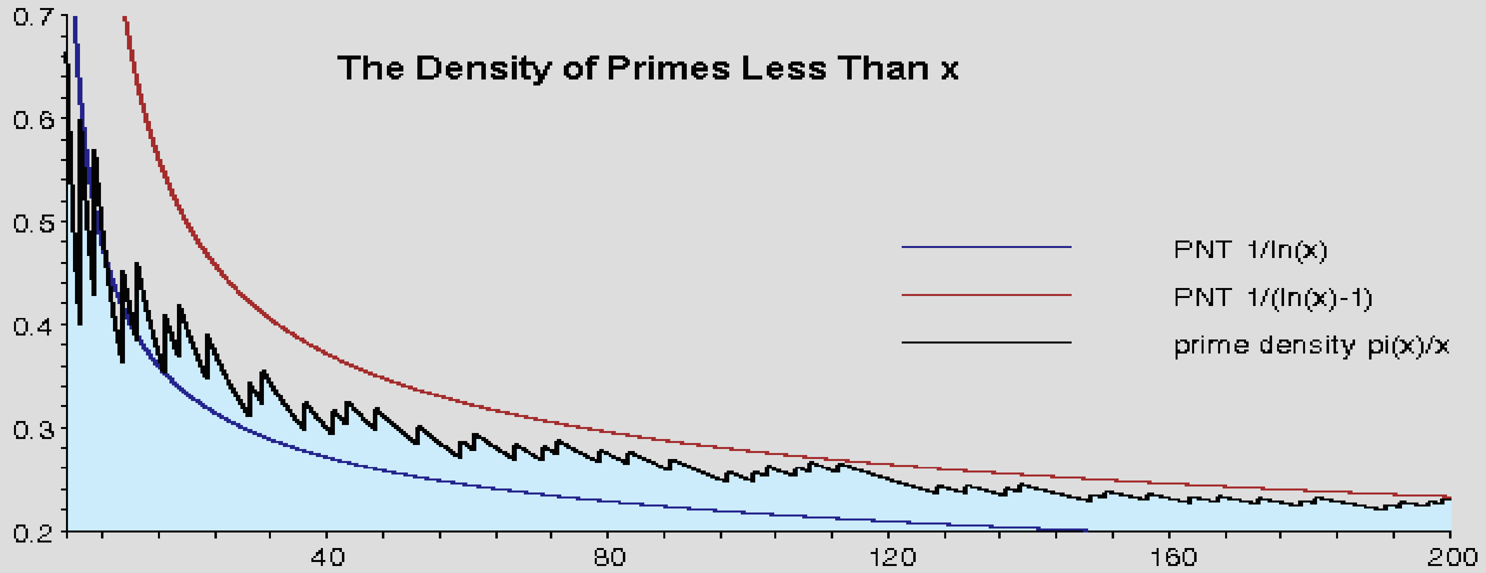
RSA (and many other applications) require large prime numbers

We need to find these efficiently (not brute force!)

The common methods are actually probabilistic (no guarantee)

Primes

First, are there actually large primes?



Density of primes around x is about 1/ln(x) (i.e. 3 per 100 when $x=10^{10}$)

Prime finding

To find them, we just make a smart guess then check if it really is prime

Smart guess:
last digit not: 2, 4, 5, 6, 8 or 0

This eliminates 60% of numbers!

Prime finding

Both of these methods use Fermat's theorem, for a prime p:

$$a^{p-1} \mod p = 1, \forall a \in \mathbb{Z}$$

So we simply check if:

2^{p-1} mod p == 1

If this is, probably prime

Prime finding

This simplistic method works surprisingly well:

Error rate less than 0.2%

(if around 512 bit range, 1 in 10^{20})

Has two major issues:

1. More accurate for large numbers
2. Carmichael numbers(e.g. 561, rare)

Prime finding

Computation time also goes up with number size

Carmichael numbers are composite, but have: $a^{p-1} \bmod p = 1$ for all a

These are quite rare though (only 255 less than 100,000,000)

Miller-Rabin primality test

Again, we will basically test Fermat's theorem but with a twist

We let: n-1 = u * 2^t, for some u and t

Then compute: $a^{n-1} \mod n == 1$

As: $a^{u \cdot 2^t} \mod n == 1$

(more efficient, as we can square it)

Miller-Rabin primality test

Witness(a, n)

find (t,u) such that $t \geq 1$ and $n-1=u*2^t$

$x_0 = a^u \bmod n$

for i = 1 to t

    $x_i = x^2_{i-1} \bmod n$

    if $x_i == 1$ and $x_{i-1} \neq 1$ and $x_{i-1} \neq n-1$

        return true

if $x_i \neq 1$

    return true

return false

Miller-Rabin primality test

If Witness returns true, the number is composite

If Witness returns false, there is a 50% probability that it is a prime

Thus testing “s” different values of “a” (range 0 to n-1) gives error 2^{-s}

Composites

To find composites of n takes (we think) O(sqrt(n))

This is the same asymptotic running time as brute force

(i.e. n%2 ==0, n%3==0, ...)

Composites

Many security systems depend on the fact that factoring nubmers is (we think) a hard problem

In RSA, if you could factor n into p and q, anyone can get private key

However, no one has been able to prove that this is hard

Composites

The book does give an algorithm to compute composites

Similar to security hashing:
(finding hash collision)

Still O(sqrt(n))
(smaller coefficient)