More Number Theory

Note Title 12/4/2013

Announcements

- No office hours tomorrow
  instead Friday 8:30-10
- No class Friday
- Last HW is up - optional
  is due at start of class
- Review Monday
- Office hours: next Wed. morning

Recap: Modular arithmetic

$\mathbb{Z}_n = \{0, \ldots, n-1\}$
(think remainders)

- Additive inverses:
  y is x's inverse if
  $(x+y) \equiv 0 \mod n$
  $5 \mod 11 \rightarrow 6$
- Multiplicative inverses:
  z^{-1} is z's inverse mod n
  if $z \cdot z^{-1} \equiv 1 \mod n$

Thm: An element x in $\mathbb{Z}_n$ has a multaplicative inverse

$\Longleftrightarrow$ $\gcd(x, n) = 1$

$\hookrightarrow$ relatively prime

Corollary: Let $x > 0$ be in $\mathbb{Z}_n$ s.t. $\gcd(x, n) = 1$. Then

$$\mathbb{Z}_n = \{ix : i = 0, \ldots, n-1\}$$

Ex: $\mathbb{Z}_{11}$, $x = 2$: $1 \cdot 2,\ 2 \cdot 2,\ \underset{2 \cdot 6}{3 \cdot 2}, \ldots$

$2, 4, 6, 8, 10, \underset{=}{1}, 3, 5, 7, 9$

Fermat's Little Thm

If p is prime, & x an integer such that $x \bmod p \neq 0$.

Then $x^{p-1} \equiv 1 \mod p$

Euler's

Totient Function: $\phi(n) = \#$ of relatively prime integers $\leq n$

$\phi(p) = p-1$

$\phi(p \cdot q) = (p-1)(q-1)$

Euler's thm: n a positive integer, & x an integer s.t. $\gcd(x,n)=1$.

Then $x^{\phi(n)} \equiv 1 \mod n$.

note: if n is prime, then $x^{\phi(n)} = x^{p-1}$

Modular Inverses

(a, b)
$\hookrightarrow (b, a \bmod b)$

Assume $\gcd(x, n) = 1$. How to compute $x^{-1} \in \mathbb{Z}_n$?

Well, remember Euclidean algorithm:

$$\gcd(x, n) = 1 = ix + jn$$

$\Rightarrow$ i is x's inverse mod n!

$$(ix + \cancel{jn}_{0}) \bmod n = 1$$

(a, b)

$\hookrightarrow (b, a \bmod b)$

Extended Euclidean Alg:

Know $d = \gcd(a, b)$ computed by Euc. algorithm:

Let $q = \underbrace{a \bmod b}$, and r be integer s.t. $a = rb + q$

Euclid's algorithm repeatedly does:

$$d = \gcd(a, b) = \gcd(b, \underset{\substack{\parallel \\ a \bmod b}}{q})$$

$GCD(b, \overset{q}{\overbrace{a \bmod b}})$

We want to augment the algorithm, so that each call on b, q also returns k and l where:

$$d = \underline{k \cdot b + l \cdot q}$$

Why? had $q = a \bmod b$

and $a = rb + q$

$\Leftrightarrow q = a - rb$

Plug in: $d = kb + lq$

Plug in: $d = kb + \ell q$

$$= kb + \ell(a - rb)$$

$$= \underbrace{\ell}_{= \boxed{i}} \cdot a + \underbrace{(k - \ell r)}_{= j} \cdot b$$

a's inverse mod b

ExtendedEuclidGCD(a, b):

If $b = 0$
    return $(a, 1, 0)$ $a = 1 \cdot a + 0 \cdot b$
$q \leftarrow a \bmod b$
$r \leftarrow$ integer s.t. $a = rb + q$
$(d, k, l) \leftarrow$ ExtendedEuclidGCD(b, q)
return $(d, \underbrace{l}_{i}, \underbrace{k - l \cdot r}_{j})$

Runtime: $O(\log n)$

Now: back to RSA

Steps:
- Select two large primes, p & q
- Let $n = p \cdot q$
  - ↳ so $\phi(n) = (p-1)(q-1)$
- Select e & d so that
  - e and $\phi(n)$ are relatively prime
  - $ed \equiv 1 \mod \phi(n)$

← Extended Euclid GCD

(In practice, e is chosen randomly or is often just $= 3, 17,$ or 65537)

So: • e & d are multaplicative inverses mod $\phi(n)$

How to compute?

previous alg.

I know e and $\phi(n)$.

Now: • (e, n) is <u>public</u> key

• d is private key (so not shared or posted)

↳ & p & q

• Why can't attacker just compute d themselves?

Need $\phi(n)$ to run EucGCD

Encrypting:

Take a message M, with $0 < M < n$.

(If longer, just split up.)

Then $C \leftarrow M^e \bmod n$

which can be easily calculated just with the public key.

Decrypting: $C = M^e \bmod n$ arrives.

Compute $C^d \bmod n$

$$C^d \bmod n = (M^e)^d \bmod n$$
$$= M^{ed} \bmod n$$

Goal: convince you that $M^{ed} = M \bmod n$

Know: $ed \equiv 1 \bmod \phi(n)$

Decrypting cont: $ed \equiv 1 \bmod \phi(n)$

$$\Rightarrow ed = k\phi(n) + 1$$

$$M^{ed} \bmod n \equiv M^{k\phi(n)+1} \bmod n$$

$$\equiv M^{k\phi(n)} \cdot M \bmod n$$

$$\equiv \left(M^{\phi(n)}\right)^k \cdot M \bmod n$$

$$\equiv (1)^k \cdot M \bmod n \equiv M$$

So: - easy for me to decrypt since I know d.

But without d, stuck!

So attacker must find d.

Problem: d is e's inverse mod $\phi(n)$

Attacker knows n, not $\phi(n)$.

How to find $\phi(n)$?

- figure out p & q
- compute directly

So this whole thing is secure as long as the attacker can't find p and q (or $\phi(n)$).

Factoring (find p and q) is <u>not</u> NP-Hard.

(In fact, no proof of <u>any</u> kind of hardness.)

Best algorithms are subexponential but still slow: Number field Seive

$$O\left(e^{\left(\frac{64}{9}\log n\right)^{1/3}(\log\log n)^{2/3}}\right)$$

Runtime of RSA (encrypting & decrypting):

Taking exponents mod n.

size input = $O(\log n)$

Setup:

Generating keys is most of it.

How?

Generating #s (p + q)

↳ Primality testing

Euclid's algorithm