Number Theory Part V

Integer factoring, discrete log algorithms, standard parameters

Integer Factoring

- Factoring trial-and-error method: $\forall\ p \in [2,..,\lfloor\sqrt{N}\rfloor]$, check if p divides N
- Complexity: $O(\sqrt{N}\ \mathrm{polylog}(N))$, N input integer — exponential-time algorithm*
- Eh...? "*exponential*"... how so?!

- (Slightly) better ones:
  - Pollard's (p-1) algorithm
  - Pollard's rho algorithm
  - Quadratic-sieve, General number-sieve

*: i.e., exponential in the size of N. Size of input is 2^n, n = no. of bits in N, so $O(2^{n/2} \log (2^n)^k)$, for some $k \geq 1$

Integer Factoring

- Alas! “better” ones still super-polynomial :-(
- …But some sub-exponential :-)

- Best known — General number-sieve
  - For value N, complexity $O(2^{O(\log N)^{1/3}} \bullet (\log(\log N))^{2/3})$*
  - Used to factor RSA-768 ≈ 2 days

- What about quantum algorithms (Shor’s method — polynomial time)?
- Err… forget quantum

*: Sub-exponential in the size of N, n bits. Work it out!

Pollard's p-1 Algorithm

- Let N = pq; p, q prime (so p-1,q-1 can't be prime), assume p-1 has small factors
- Pick $B \in Z+$, s.t., $(p-1) \mid B$, and $(q-1) \nmid B$, let $B=\gamma(p-1) + 0$, for some γ
- Let $B = \prod_{i=1}^{k} p_i^{\lfloor n/\log p_i \rfloor}$, for some k; p_i is the i^{th} prime, n = |p| in bits.[1]
- Why? Because $p_i^{\lfloor n/\log p_i \rfloor}$ is the largest power of p_i that divides p-1.[1]

1: Known result in abstract algebra. Don't believe me? Refer to any AA text, e.g., Dummit and Foote, 2013. ISBN 978-0471433347

Pollard's p-1

- Re-writing, let $p-1 = \prod^{k}_{i=1} p_i^{e_i}$, for some $e_i \geq 0$
- Then, $p-1 \mid B$, and $q-1 \nmid B$[1]
- Choosing a larger k increases B (and running-time)
- Find $x \leftarrow Z_N^*$, and compute $y = x^B - 1 \bmod N$
- From CRT, $1 \leftrightarrow (1,1)$ for any $N > 1$

1: i.e., if the largest prime factor of $p-1 < p_k$, and $q-1$ has any prime factor $> p_k$

Pollard's p-1

- So,

$y = x^B - 1 \bmod N \leftrightarrow (x_p, x_q)^B - (1,1)$

$= (x_p^B - 1 \bmod p,\ x_q^B - 1 \bmod q)$

$= ((x_p^{p-1})^\gamma - 1 \bmod p,\ x_q^B - 1 \bmod q)$

$= (0, [x_q^B - 1 \bmod q])$

Assume $x_q^B - 1 \bmod q \neq 0$[1]

If so, $y = 0 \bmod p$, but $y \neq 0 \bmod q$, i.e., $p \mid y$, but $q \nmid y$, so, $\gcd(y, N) = p$

1: Only happens if we end up with x_q as a generator of Z_q^* — if so, go back and pick another x

Pollard’s p-1

- Pollard’s p-1 factoring algorithm:

p-1 Algorithm (N) —> p /* returns one factor */

find x <— Z_N^*

/* compute B as in slide 4 */

compute y = x^B -1 mod N

let p = gcd(y,N)

if p $\notin$ {1,N}, return p /* Avoid trivial factors */

Pollard's p-1

- Works only if p-1 *or* q-1 have small factors
- Unlikely either have small factors in most crypto applications…
- Complexity exponential in size of N: $O(B \bullet \log B \bullet (\log^2 N))$
- Better factoring algorithms available

Pollard's Rho Algorithm

- General-purpose factoring algorithm

- Idea: Given N, find *distinct* $x, x' \in Z_N^*$, s.t., $x \equiv x' \bmod p$, for some $p > 1$

- Point is to find a "good" pair (x,x'), s.t., $\gcd(|x-x'|, N) = p$

- Where will such a good pair come from?

Pollard's Rho

- Pick $(x^{(1)}, \dots, x^{(k)}) \in Z_N^*$, $k = 2^{n/2}$ (known result in AA; refer Dummit and Foote for proof)
- Applying CRT, $(x^{(1)}_p, x^{(1)}_q), \dots (x^{(k)}_p, x^{(k)}_q)$
- $x^{(i)}_p = [x^{(i)} \bmod p]$, and $x^{(i)}_q = [x^{(i)} \bmod q]$
- Each $x^{(i)}_p$ $x^{(i)}_q$ uniformly distributed in Z_p^*, Z_q^*
- Forget q, just consider p…(why?)[1]

1: All we want is 1 factor of N. Then just do q = N/p

Pollard's Rho

- Detour – Birthday paradox:
  - For an $N \in Z+$, given random, uniformly chosen q elements $(x_1,\ldots x_q) \in \{1,..N\}$, probability that $\exists$ i, j, s.t., $i \neq j$, $x_i = x_j$ is at most $(q^2/2N)$
  - Pr [collision between x_i, x_j] $\leq q^2/2N$
- For our purpose: $\exists$ distinct i, j, s.t., $x^{(i)}{}_p = x^{(j)}{}_p$, with high probability, but $x^{(i)} \neq x^{(j)}$ with high probability

Pollard's Rho

- Great! Now we've found a "good pair": $x^{(i)}{}_p = x^{(j)}{}_p$, or $x^{(i)} \bmod p = x^{(j)} \bmod p$
- Recollect, we wanted a "good" pair: (x,x'), s.t., $\gcd(|x-x'|, N) = p$[1]
- Just find $\gcd(|x^{(i)}-x^{(j)}|,N) \longrightarrow p$
- Once p found, trivial to find q

1: This will always yield p, since $x \neq x' \bmod N$

Pollard's Rho

- Pollard's rho factoring algorithm:

rho Algorithm (N) —> p /* returns one factor */

find $x^{(0)} \leftarrow Z_N^*$

for i = 1 to $2^{n/2}$ do

$x^{(i)} = F(x^{(i-1)})$

p = gcd $(| x^{(i-1)}-x^{(i)}|, N)$

if $p \notin \{1,N\}$, return p /* Avoid trivial factors */

What is "F"?

Pollard's Rho

- F: $Z_N^* \longrightarrow Z_N^*$ needs to have property:
  - If $x \bmod p = x' \bmod p$, then $F(x) \bmod p = F(x') \bmod p$

- Standard choice: $F(x) = (x^2 + 1) \bmod N$; any polynomial has this property

- Complexity: $O(N^{1/4}\,\mathrm{polylog}(N))$… still exponential in size of N

Quadratic Sieve

- Based on quadratic residues mod N
- $z \in Z_N^*$ is a *quadratic residue* mod N if there is an $x \in Z_N^*$, s.t., x^2 mod N = z mod N
- x is a *square root* of z mod N
- Known result: if N = pq; p, q primes > 2, p ≠ q, every quadratic residue mod N has exactly 4 square roots

Quadratic Sieve

- Given x, y, $x^2 \bmod N = y^2 \bmod N$, and $x \neq \pm y \bmod N$, $\gcd(x-y, N) = p$, $p \notin \{1,N\}$ is a factor of N

- Idea: Find a x, y, s.t., $x^2 \bmod N = y^2 \bmod N$, and $x \neq \pm y \bmod N$

- Naive method[1]:
  - Generate an $x \in Z_N^*$,
  - Find $q = x^2 \bmod N$
  - Check if $q = y^2$ for some $y \in Z+$

1: Forms the basis of Fermat's primality test

Quadratic Sieve

- Too slow (usually don't end up with a $q=y^2$)
- Better method idea: generate sequence of q's: $q_1=x_1^2 \bmod N, \ldots, q_l=x_l^2 \bmod N$
- Identify a subset of q_i's whose *product* is y^2 (better chances than naive method): "smooth" integers[1]
- Quadratic sieve best known until early 1990s (still used for $n < 300$ bits $\approx$ 100 decimal digits)
- Second-best method

[1] Smooth integers factor completely into small prime factors

General Number Field Sieve (GNFS)

- State-of-the-art for integer factoring
- Quadratic sieve best known until early 1990s (still used for $n < 300$ bits, $N = 2^n$)
- Idea: main bottleneck is finding smooth q_i's – $2^{n/2}$, $n = |N|$
- Speed-up time for finding smooth integers – actually achieves in sub-exp. time

Computing Discrete Log

- General case: Z_q^* for any $q > 1$
  - Pohlig-Hellman algorithm (composite N)
  - Baby-step giant-step algorithm
  - Pollard's rho algorithm

- Special case: Z_q^*, q prime
  - Index calculus algorithm
  - Number field sieve

- None in polynomial-time, but best is sub-exponential

Computing Discrete Log

- Pohlig-Hellman:
  - Works over all-order groups; best performance for composite-order
  - Complexity dominated by size of largest subgroup of prime order, p
  - $O(\sup\{\sqrt{p}\} \bullet \text{polylog } q)$
  - Worst case (when q is prime): $O(\sqrt{q} \text{ polylog } q)$
  - In terms of sizes: $O(\sup\{2^{p'/2}\} (\log 2^{q'})^k)$, $k \geq 1$, $p' = |p|$, $q' = |q|$

Computing Discrete Log

- BSGS:
  - Works for all-order groups
  - Basic idea: say, $G = \{g^0, g^1, g^2, \ldots, g^{q-1}\}$
  - Given $v = g^h \bmod q$, we're trying to find $h \in \{1\ldots q-1\}$
  - Divide G into intervals ("giant" step), I, $|I|=t$
  - Compute $g^1 \ldots g^t$ for each I ("baby" step), efficiency gains by sorting…
  - Complexity: $O(\sqrt{q} \text{ polylog } q)$, $|G| = q$

Other methods

- Index Calculus
  - Works for Z_q^*, q is prime
  - Sub-exponential in |q|: $2^{O(\sqrt{\log q} \cdot \log(\log q))}$

- GNFS for DL:
  - Best known for Z_q^*, q is prime
  - Sub-exponential in |q|: $2^{O((\log q)^{1/3} \cdot \log(\log q)^{2/3})}$

Recommended Lengths

Modulus (N) Length (bits)	Symmetric Key Equivalent (bits)	Discrete Log (bits): (p-1) = $\|Z_p^*\|$, Order-q subgroup of Z_p^*
2048	112	p = 2048, q = 224
3072	128	p = 3072, q = 256
7680	192	p = 7680, q = 384
15360	256	p = 15360, q = 512