Chapter 5: Factoring Algorithms

Math 495, Fall 2008

Hope College

October, 2008

Square Roots Modulo n

- Of interest here is how many solutions y the equation $y^2 \equiv a \bmod n$ has, where $\gcd(a, n) = 1$.
- If n is prime, we previously saw that the answer is two if $\left(\frac{a}{n}\right) = 1$, and zero otherwise.
- If $n = p^e$, where p is prime the answer is the same.
- If $n = \prod_{i=1}^{\ell} p_i^{e_i}$, then there are 2^ℓ solutions (modulo n) iff $\left(\frac{a}{p_i}\right) = 1$ for all $i \in \{1, \ldots, \ell\}$, and zero otherwise.

More on Square Roots

- Let $x^2 \equiv y^2 \equiv a \mod n$, where $\gcd(a, n) = 1$.
- Define $z = xy^{-1} \mod n$.
- It is not hard to see that $z^2 \equiv 1 \mod n$.
- On the other hand, notice that if $z^2 \equiv 1 \mod n$, then $(xz)^2 \equiv x^2 \mod n$ for any x.
- Thus, if $x^2 \equiv a \mod n$, then $(xz)^2 \equiv x^2 \equiv a \mod n$, where z is any square root of 1 modulo n.
- Thus, the square roots of a modulo n can be computed by finding 1 square root of a modulo n, and multiplying it by each of the 2^ℓ square roots of 1 modulo n.

Factoring Algorithms

- **Trial division**
- **Pollard's** $p-1$ **Algortihm**
- **Pollard's rho Algorithm**
- **Random Squares Algorithm**
- Williams's $p+1$ algorithm
- Continued fraction algorithm
- Quadratic Sieve
- Elliptic Curve Factoring Algorithm
- Number Field Sieve
- **Shor's Algorithm**

Trial Division

- Recall that if n is composite, it has a factor less than $\sqrt{n}$.
- Thus to factor n, we can divide n by every number between 2 and $\sqrt{n}$ (or just the primes) and see if the remainder is 0.
- This algorithm takes $\Omega(\sqrt{n})$ operations.
- Recall that Ω is a lower bound.
- Unfortunately, $\sqrt{n}$ is exponential in $\log n$, making this algorithm totally impractical for large numbers.

Pollard $p-1$ Algorithm

- Let p be a factor of n.
- Assume that all prime factors of $p-1$ are less than B.
- Then clearly $(p-1)|B!$, implying $B! = (p-1)r$ for some r.
- Let $a \equiv 2^{B!} \bmod n$
- Then $a \equiv 2^{B!} \equiv 2^{(p-1)r} \equiv 1^r \equiv 1 \pmod{p}$
- So we have an a such that $a \equiv 1 \bmod p$.
- Therefore $a-1 \equiv 0 \bmod p$.
- Thus, if $a \neq 1$, then $a-1$ is a multiple of p.
- Then $d = \gcd(a-1, n)$ is a factor of n.

Pollard $p-1$ Algorithm Analysis

- To compute $2^{B!}$, we need $B-1$ modular exponentiations.
- Each exponentiation requires $O(\log B)$ modular multiplication operations.
- Each modular multiplication requires $O((\log n)^2)$ operations.
- The gcd takes $O((\log n)^3)$
- Thus, the algorithm requires time $O((B-1)\log B(\log n)^2 + (\log n)^3)$ operations.
- If $B = O((\log n)^i)$ for some constant i, the algorithm is polynomial-time (in $\log n$), but the chance of success is small.
- To increase the chance of success, B may need to be as high as $\sqrt{n}$, at which point the algorithm is no better than trial division.

Pollard $p-1$ Algorithm can fail

- The textbook claims that the algorithm is guaranteed to be successful if B is chosen to be around $\sqrt{n}$
- Let's take a look at an example in Maple.
- Notice that if B is greater than all of the prime factors of $p-1$ and $q-1$, then it is possible that $a^{B!} \equiv 1 \mod n$.
- There are various ways to fix this, but we will not discuss them at length.

Pollard $p-1$ algorithm and RSA

- Let $n = pq$, for primes p and q.
- We can choose p and q such that Pollard $p-1$ Algorithm will be totally ineffective.
- Consider primes p and q such that

$$p = 2p_1 + 1$$

$$q = 2q_1 + 1$$

  where p_1 and q_1 are prime.
- We would need to choose $B \approx p_1 \approx \sqrt{n}/4$ for the algorithm to succeed.

Pre-Pollard Rho Algorithm

- Let $n = pq$ as usual.
- Let $x, y \in \mathbb{Z}_n$ with $x \neq y$ and $x \equiv y \bmod p$.
- Then $p \leq \gcd(x - y, n) < n$, yielding a factor of n.
- Of course we do not know p, so we cannot tell whether or not $x \equiv y \bmod p$.
- But we can pick a bunch of distinct random numbers, and compute $\gcd(x - y, n)$ for each $x \neq y$ until we get a factor.
- The *Birthday paradox* implies that if we have about $1.17\sqrt{p}$ numbers, there is a 50% chance of a collision.
- Unfortunately, we cannot test for a collision, but only compute the gcd for every pair in the set.
- There are $\binom{1.17\sqrt{p}}{2} > p/2$ pairs.
- Thus we need to consider about $p/2 \approx \sqrt{n}$ pairs (assuming p and q are close to the same length).

Pollard Rho Insights

- Let f be some polynomial with integer coefficients.
- Define a sequence of numbers by letting $x_1 \in \mathbb{Z}_n$, and defining $x_i = f(x_{i-1})$ mod n for $i > 1$.
- Pick an integer m and define $X = \{x_1, \ldots, x_m\}$.
- We assume that X is a random sampling from $\mathbb{Z}_n$.
- We do not want to compute $\gcd(x_i - x_j, n)$ for every pair of numbers from X.
- We will look at an example in Maple and this will give us some insight into how to avoid looking at all pairs, and demonstrate why this is called the Pollard *Rho* Algorithm.

Pollard Rho Algorithm

- $f = x^2 + a$ for some $a \neq 0, -2$.
- Define a sequence of numbers by letting $x_1 \in \mathbb{Z}_n$, and defining $x_i = f(x_{i-1}) \bmod n$.
- The Pollard Rho algorithm computes $p = \gcd(x_{2i} - x_i, n)$ for $i = 1, \ldots,$ until $p \neq 1$.
- Then $p = n$ or is a non-trivial factor of n.
- We'll look at an example in Maple.
- What if it fails?
  - Choose new value for x_1 and try again
  - Choose a new value for a and try again
- Expected number of iterations is about $\sqrt{p}$,
- Running time is about $O(\sqrt{p}) \approx O(\sqrt[4]{n}) = O(n^{1/4})$.
- Why can we restrict our attention to pairs x_i and x_{2i}?
- Start by considering the diagram on the board based on the Maple example.

Pollard Rho Algorithm: Why it works

- Let $x_i \equiv x_j \bmod p$, where $i < j$.
- Then $f(x_i) \equiv f(x_j) \bmod p$.
- Further, $x_{k+1} \bmod p = (f(x_k) \bmod n) \bmod p = f(x_k) \bmod p$
- Therefore $x_{i+1} \equiv x_{j+1} \bmod p$
- In general then, $x_{i+\delta} \equiv x_{j+\delta} \bmod p$
- Let $\ell = j - i$. Then notice that $x_{i'} \equiv x_{j'} \bmod p$ as long as $j' > i' \geq i$ and $j' - i' \equiv 0 \bmod \ell$. (See diagram on board)
- Let x_i, x_j be the first pair with $i < j$ such that $x_i \equiv x_j \bmod p$.
- Let k be some integer with $i \leq k < j$ that is divisible by ℓ. (why is there one?)
- Then $x_k \equiv x_{2k} \bmod p$, since $k \geq i$, and k and $2k$ are some multiple of ℓ apart.
- Thus, we can restrict our attention to pairs x_k, x_{2k}.

Dixon's Random Squares Algorithm

- Suppose $x^2 \equiv y^2 \bmod n$, where $x \neq \pm y \bmod n$.
- Then $n|(x-y)(x+y)$, so we have two factors of n: $\gcd(x-y, n)$ and $\gcd(x+y, n)$.
- The only problem: How do we find such values?
- Let B be a set of prime numbers (the smallest k) and -1.
- Compute a set of values Z such that for each $z \in Z$, all factor of $z^2 \bmod n$ occur in the set B.
- Find a subset of $Y \subseteq Z$ such that each number in B occurs an even number of times as a factor of $z^2 \bmod n$.
- How do we pick the values of z?
- Choose numbers like $j + \left\lceil \sqrt{kn} \right\rceil$ and $\left\lfloor \sqrt{kn} \right\rfloor$, for small values of j and k.
- Why do these values work well?

Dixon's Algorithm Example

- We will see an example of factoring 30049 using Maple.
- You can read your textbook for the details.

Dixon's Algorithm Analysis

- If we have at least as many congruences as we have elements in the base, there is a linear dependence.
- There is at least a 50% chance that a given solution will yield a factorization.
- Clearly there is a trade-off here: larger factor base means higher chance of success, but also more computation and storage.
- If $n \approx 2^r$, then a good size for a factor base is $2^{\sqrt{r \log_2 r}}$.
- Waiving hands a bit, throwing in some fairy dust and an act of God, we arrive at the fact that if we choose a factor base of optimal size, the algorithm has an expected running time of $O(e^{(1+o(1))\sqrt{\ln n \ln \ln n}})$

Shor's Algorithm

- We will look at Shor's algorithm when we discuss quantum cryptography and algorithms.
- For now, I'll just say that is it a polynomial-time factoring algorithm.
- What are the implications of this?
- What aren't the implications of this?
- We will briefly discuss P, NP, NP-Complete, and what these have to do with cryptography and factoring numbers.