Pollard's DL Algorithm

Exposition by William Gasarch

1 Mathematics We Will Need

Lemma 1.1 *If $n = \frac{x}{y}$ is an integer and p is a prime that divides x but not y then p divides n.*

Proof: Factor both x and y. There will be a factor of p in x but not in y. When you reduce to lowest terms all of the prime factors of y will go away. Some of the prime factors of x will go away, but not p. Hence p will remain. This yields a factorization of x where p is one of the factors. ∎

The following lemma you should know from when you studied combinatorics, though I may go over the proof in class.

Lemma 1.2 *The number of ways to choose b items from a items is $\binom{a}{b} = \frac{a!}{b!(a-b)!}$.*

Lemma 1.3 *For all primes p, for all $1 \le y \le p-1$ p divides $\binom{p}{y}$.*

Proof: $\binom{p}{y} = \frac{p!}{y!(p-y)!}$ is an integer where p divides the numerator but not the denominator. By Lemma 1.1 p divides $\binom{p}{y}$. ∎

The following you have surely seen. I may prove it in class

Lemma 1.4 *Let $n \in \mathsf{N}$. Then $(x+y)^n = \sum_{i=0}^{n} \binom{n}{i} x^i y^{n-i}$.*

Lemma 1.5 *Let p be a prime and $n \in \mathsf{N}$. Then $n^p \equiv n \pmod{p}$.*

Proof: We prove this by induction on n.
Base case: If $n = 1$ then $n^p = 1^p = n \pmod{p}$.
Induction Hypothesis: Assume that $n^p \equiv 1 \pmod{p}$ and that $n+1 \le p-1$.
Induction Step:

$$(n+1)^p \equiv \sum_{i=0}^{p} \binom{p}{i} n^i 1^{p-i} = \sum_{i=0}^{p} \binom{p}{i} n^i = 1 \times n^0 + \sum_{i=1}^{p-1} \binom{p}{i} + 1 \times n^p.$$

By Lemma 1.3 all of the terms in $\sum_{i=1}^{p-1} \binom{p}{i}$ are $\equiv 0 \pmod{p}$. Hence we have

$$(n+1)^p \equiv \sum_{i=0}^{p} \binom{p}{i} n^i 1^{p-i} = 1 + n^p.$$

By the induction hypothesis $n^p \equiv n \pmod{p}$, so we have $(n+1)^p \equiv n+1 \pmod{p}$.
∎

Lemma 1.6 *If $1 \le n \le p-1$ and p is prime then $n^{p-1} \equiv 1 \pmod{p}$.*

Proof: By Lemma 1.5 $n^p \equiv n \pmod{p}$. Hence there is a k such that

$$n^p = n + kp.$$

Divide by n to obtain

$$n^{p-1} = 1 + \frac{kp}{n}.$$

Since $\frac{kp}{n} = n^{p-1} - 1$, $\frac{kp}{n}$ is an integer. Since $n \le p-1$, p does not divide the denominator n, though p clearly divides the numerator kp. Hence we can apply Lemma 1.1 and conclude that p divides $\frac{kp}{n}$. Hence

$$n^{p-1} \equiv 1 \pmod{p}.$$

▮

Lemma 1.7 *Let p be a prime. Then $a^n \equiv a^{n \pmod{p-1}} \pmod{p}$.*

Proof:
Let $n \equiv n' \pmod{p-1}$ where $0 \le n' \le p-1$. Hence $n = n' + k(p-1)$ for some k. Then

$$a^n = a^{n'+k(p-1)} = a^{n'} \times a^{k(p-1)} = a^{n'} \times (a^{p-1})^k$$

By Lemma 1.6 $a^{p-1} \equiv 1 \pmod{p}$. Hence we have $a^n \equiv a^{n'} \pmod{p}$. ▮

2 An $O(\sqrt{p})$ Time, $O(\sqrt{p})$ Space Algorithm for Discrete Log

Definition 2.1 The *Discrete Log Problem* is as follows. Let p be a prime and g be a generator. These are considered parameters. Given $y \in \{1, \ldots, p-1\}$ find x such that $y = g^x$.

Throughout this paper p is a prime and g is a generator for it.

SO, given p, y we want to find x such that $y \equiv g^x \pmod{p}$. What if we find a, b such that

$$y^a \equiv g^b \pmod{p}$$

Is this helpful?

Assume we knew $a^{-1} \bmod p-1$ (YES that is a $p-1$). Then we can raise both sides to the a^{-1} power to get

$$(y^a)^{a^{-1}} \equiv g^{ba^{-1}} \pmod p$$

KEY: By Lemma 1.7 on both sides we can reduce the exponent mod $p-1$. Hence we have

$y \equiv g^{ba^{-1}} \pmod p$.

SO, we will have found the discrete log! Its just $ba^{-1} \pmod{p-1}$.

RECAP: Assume that taking inverses mod $p-1$ is easy (this is true). Then the problem of finding the discrete log of y can be solved if we find the discrete log of some power of y.

PLAN: Given p, g, y we LOOK FOR a, b such that $y^a \equiv g^b \pmod p$. Can we find these quickly? (At least more quickly than $O(p)$ steps). We can!

Theorem 2.2 *(The birthday paradox) Let s, p be such that $s \ll p$. Assume you have a bin of n balls numbered $\{1, \ldots, p\}$. You pick a ball at random, record its number, throw it back. You do this $s+1$ times. Then the probability that there is a pair of balls that you picked out that have the same number is $\geq 1 - e^{-s^2/2p}$.*

Proof: The total number of ways that the balls can be picked is p^s. The number of ways that they are all different is $p(p-1)(p-2)\cdots(p-s)$. Hence the probability that the balls are all distinct is

$$\frac{p(p-1)(p-2)\cdots(p-s)}{p^k} = \frac{p}{p}\frac{p-1}{n}\frac{p-2}{p}\cdots\frac{p-s}{p} =$$

which is

$$1 \times \left(1-\frac{1}{p}\right)\left(1-\frac{2}{p}\right)\cdots\left(1-\frac{k}{p}\right)$$

Since $s \ll p$ we can, for $1 \leq i \leq s$, approximate $\left(1-\frac{i}{p}\right)$ by $e^{-i/p}$. Hence the probability that all the balls are distinct is approximately

$$1 \times \left(1-\frac{1}{p}\right)\left(1-\frac{2}{p}\right)\cdots\left(1-\frac{s-1}{p}\right) = e^{-1/p}e^{-2/p}\cdots e^{-s/p} = e^{-(1+2+\cdots+s)/p} \sim e^{-s^2/2p}$$

∎

THIS IS AS FAR AS I GOT ON JAN 16.

Theorem 2.3 *There is an algorithm for discrete log that takes $O(\sqrt{p})$ time and $O(\sqrt{p})$ space.*

Proof: We assume that p, g are known. We will be given y as input. The algorithm essentially looks for a, b such that $y^a \equiv g^b \pmod p$. We will keep track of a set of triplets $(c, d, y^c g^d)$. If we find two triples $(c, d, y^c g^d)$ and $(c', d', y^{c'} g^{d'})$ where $y^c g^d \equiv y^{c'} g^{d'}$ then note that we obtain $y^{c-c'} \equiv g^{d'-d}$. We will then let $a = c - c'$ and $b = d - d'$.

We proceed formally.

We will need a data structure to hold a set X. We leave it unspecified for now but keep in mind that it will be able to do $INSERT$ and $FIND$.

1. Input(y)
2. $DONE = FALSE$, $X = \emptyset$, $j = 1$. X will be the set of triples.
3. While NOT DONE do the following
   (a) Pick random pair $(c_j, d_j) \in \{1, \ldots, p-1\} \times \{1, \ldots, p-1\}$. (From now on whenever c_i, c_j, d_i, d_j are mentioned it is assumed they are in $\{1, \ldots, p-1\}$ and that any math with them is $\pmod{p-1}$.
   (b) FIND if $y^{c_j} g^{d_j}$ is in X. (This will take one FIND.) If YES then you've found an $i < j$ such that $y^{c_i} g^{d_i} = y^{c_j} g^{d_j}$ If such a j is found then set $a = c_j - c_i$, $b = d_i - d_j$. If a has in inverse mod $p - 1$ then set DONE=YES; Else set $i = i + 1$ and INSERT $(c_j, d_j, y^{c_j} g^{d_j}$ into X. (This will take one INSERT.)
4. (If you got to this step then have a, b such that $y^a = g^b$ and a has an inverse mod $p - 1$.) Find $a^{-1} \pmod{p-1}$. Output the ANSWER: ba^{-1}.

How many iterations do we expect? Let s be some number of iterations. The numbers $y^{c_j} g^{d_j}$ can be viewed as RANDOM NUMBERS FROM $\{1, \ldots, p\}$. And we hope that two of the numbers match. This is EXACTLY the problem from Theorem 2.2: we are picking s numbers from $\{1, \ldots, p\}$. By that theorem the probability that some two match is $1 - e^{-s^2/2p}$. So after $s = c\sqrt{p}$ iterations (we'll pick c later) the probability of a match is $1 - e^{-c}$. If $c = 4$ then this is 0.55. Thus using $c = 4$ we will expect to find a collision about half the time. Hence the expected number of iterations is bonded by $4\sqrt{p}$. (A more sophisticated analysis shows that the expected number of iterations is $\sqrt{\frac{\pi}{2}}\sqrt{p} \sim 1.25\sqrt{p}$.)

In iteration j we do one FIND and one INSERT We assume that our data structure can do both in the same time and call that time t. operations. Hence the algorithm takes $O(\sqrt{p} \times t)$.

There are various tree data structures that can do FIND and INSERT in $O(\log p)$ steps where p is the max number of elements in the data structure. There are hashing schemes that can do both operations in $O(1)$ steps. Hence the algorithm takes $O(\sqrt{p})$ steps. ∎

The importance of this algorithm is that for those using Diffie-Helman you might THINK that you need to take p to be large. We know know that that's not good enough— we need $\sqrt{p}$ to be large.

DO NOT READ PAST THIS POINT, STILL WORKING ON IT

3 An $O(\sqrt{p})$ Time, $O(\log p)$ Space Algorithm for Discrete Log

Is the above algorithm practical? The $\sqrt{p}$ time is pretty good. But the $\sqrt{p}$ space is a real problem. Can we do better? YES- but not rigorously. What we now present works well in practice but is not on a rigorous funcation yet (*It works well in practice, but does it work in theory?*).

Do we need to maintain the entire set X?

In the above algorithm we generate the sequences at random. But its good enough if they *look random* (we are not going to define that). What if they are generated by a deterministic function f.

Lets say this *looks random*. So much so that we expect that there will be a repeat among $\{f(1), f(2), \ldots, f(c\sqrt{p})$. KEY- once there is a repeat the entire sequence loops. That is, if (say) $f(20) = f(10)$ then we have $f(21) = f(11)$, $f(22) = f(12)$, etc.

SO, NEW PROBLEM: Given a function f that is defined so that $f(i)$ depends only on $f(i-1)$ we want to find x, y such that $f(x) = f(y)$. The idiotic way is to computer *and store* $f(1), f(2), \ldots$ and check every new member against the old members (you may be able to use a fancy data structure, but it will still take a lot of time and space). We want to do this problem with very little space.

Lets do an example first with $p = 47$.

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$f(n)$	2	17	8	10	34	7	17	8	10	34	7	17	8	10	34

Imagine that we can only look at VERY FEW elements of f at one time. Could we find an x, y, with $f(x) = f(y)$. Lets look at pairs of the form $x, 2x$.

$f(1) = 1$, $f(2) = 2$

$f(2) = 17)$, $f(4) = 10$

$f(3) = 8$, $f(6) = 7$.

$f(4) = 10$, $f(8) = 8$

$f(5) = 34$, $f(10) = 34$ FOUND A REPEAT!

We state the following theorem, and provide the algorithm, but omit the proof that it works.

Theorem 3.1 *Let f be a function from N to a set S. If there exists $i < j$ such that $f(i) = f(y)$ then there exists $x \leq 3j$ such that $f(x) = f(2x)$.*

We now use this to obtain a modification of the above algorithm that uses only $O(\log p)$ space.

Theorem 3.2 *There is an algorithm for discrete log that takes $O(\sqrt{p}\log p)$ time and $O(\sqrt{p}\log p)$ space.*

Proof: Rather than generate (c_j, d_j) at random we will generate them deterministically with a function h. But we will later claim without proof that the resulting sequence *looks random*.

Choose three subsets X_1, X_2, X_3 of $\{1, \ldots, p-1\}$ of roughly the same size. We do NOT store the subsets. They are chosen so that determining which one an element is in is easy. (e.g, we could make
$X_1 = \{1, \ldots, \lfloor p/3 \rfloor\}$,
$X_2 = \{\lfloor p/3 \rfloor + 1, \ldots, \lfloor 2p/3 \rfloor\}$,
$X_3 = \{\lfloor 2p/3 \rfloor + 1, \ldots, p-1\}$.
Let (j, c, d, x) be such that $x = y^c g^d$. Then we define f as follows:

$$f(j,c,d,x) = \begin{cases} (j+1, c+1, d, yx) & \text{if } x \in X_1; \\ (j+1, 2c, 2d, x^2) & \text{if } x \in X_2; \\ (j+1, c, d+1, gx) & \text{if } x \in X_3. \end{cases} \tag{1}$$

Note that if $f(j,c,d,x) = (j+1, c', d', x')$ then $x' = y^{c'} g^{d'}$.

1. Input(y)

2. DONE=FALSE, $j = 1$.

3. Pick random pair (c_1, d_1) Store $(1, c_1, d_1, y^{c_1} g^{d_1})$. Let $x_1 = y^{c_1} g^{d_1}$. Compute and store Store $f(1, c_1, d_1, y^{c_1} g^{d_1}) = (2, c_2, d_2, x_2)$. In the future we will store a vector of the form (j, c_j, d_j, x_j) and a vector of the form $(2j, c_{2j}, d_{2j}, x_{2j})$ but nothing else.

4. While NOT DONE do the following

   (a) (We have stored (j, c_j, d_j, x_j) and $(2j, c_{2j}, d_{2j}, x_{2j})$ but nothing else. Compute $f(j, c_j, d_j, x_j) = (j+1, c_{j+1}, d_{j+1}, x_{j+1})$ and store it. DELETE (j, c_j, d_j, x_j). Compute $f(f(2j, c_{2j}, d_{2j}, x_{2j}) = (2j+2, c_{2j+2}, d_{2j+2}, x_{2j+2})$. DELETE $(2j, c_{2j}, d_{2j}, x_{2j})$.

   (b) If $x_{j+1} = x_{2j+2}$ then note that $y^{c_{j+1}} g^{d_{j+1}} = y^{c_{2j+2}} g^{d_{2j+2}}$. Set $DONE = TRUE$, $a = c_{j+1} - c_{2j+2}$, $b = d_{2j+2} - d_{j+1}$.

5. (If you got to this step then have a, b such that $y^a = g^b$.) Find $a^{-1} \pmod{p-1}$. Output the ANSWER: $ba^{-1} \pmod{p-1}$.

By the reasoning about the prior algorithm we know we can do this with $s = O(\sqrt{p})$ iterations.

Note that we only ever keep around two 4-tuples of length $O(\log p)$. Hence the space is $O(\log p)$. ∎