

CS556 Introduction to Cryptography - Prof. Ming-Deh Huang

Scribe: Iftikhar A Burhanuddin

burhanud@usc.edu

Lecture #23, #24 - November 19, 26 2002

Announcements:

1. Pari-GP assignment 2 is due on 12/06/02

Topics for today:

1. Silver-Pohlig-Hellman Method
2. Index Calculus Attack
3. Birthday Paradox Attack

Today we'll survey the available methods to solve the DL problem over finite fields.

1. Silver-Pohlig-Hellman method works when the order of the group is smooth.
2. Index Calculus attack is based on the *smoothness* trick. It is the best attack to date and gives a sub exponential time algorithm to solve the DL problem. Also next week we'll discuss a similar smoothness trick for integer factoring.
3. Generic attack on DL problem over a cyclic group G based on the *Birthday paradox*. This works unconditionally but runs in $O(\sqrt{N})$ time where $N = |G|$.

1. Silver-Pohlig-Hellman method

Given a group $G = \langle g \rangle$ and $|\langle g \rangle| = n$. Suppose n is k -smooth, that is for every prime $p \mid n$, $p \leq k$.

DL problem. Given $y \in G = \langle g \rangle$ to compute x such that $y = g^x$.

The underlying idea in this algorithm is as follows. Suppose $p^\alpha \parallel n$ then it suffices to compute $x \bmod p^n$ for all $p \mid n$ and then using CRT x can be

recovered. Now for a particular p we are interested in computing b_i where $x \equiv b_i \pmod{p^i}$ and we determine them iteratively until $i = \alpha$ and we start by computing b_0 . The fact that n is smooth lets us focus on the “small” primes dividing n one at a time.

Thm. $x \equiv b \pmod{p^i} \Leftrightarrow y^{\frac{n}{p^i}} = g^{\frac{n}{p^i}b}$.

Proof. Suppose that the RHS $y^{\frac{n}{p^i}} = g^{\frac{n}{p^i}b}$ holds. Now since $g^x = y$,

$$\begin{aligned} g^{x\frac{n}{p^i}} &= y^{\frac{n}{p^i}} = g^{\frac{n}{p^i}b} \\ \Leftrightarrow g^{\frac{n}{p^i}(x-b)} &= 1 \\ \Leftrightarrow n \mid \frac{n}{p^i}(x-b) \\ \Leftrightarrow p^i \frac{n}{p^i} \mid \frac{n}{p^i}(x-b) \\ \Leftrightarrow p^i \mid x-b \\ \Leftrightarrow x &\equiv b \pmod{p^i} \quad \diamond \end{aligned}$$

The key observation is that the group $\langle g^{\frac{n}{p^i}} \rangle$ is of order p^i since $\langle g \rangle$ is of order n and so to compute $x \equiv b \pmod{p^i}$ it is enough to solve the DL problem in the group $\langle g^{\frac{n}{p^i}} \rangle$.

The theorem tells us $x \equiv b_0 \pmod{p} \Leftrightarrow y^{\frac{n}{p}} = (g^{\frac{n}{p}})^{b_0}$. So to compute b_0 we try $b_0 = 0, 1, \dots, p-1$ and check when equality holds between $y^{\frac{n}{p}}$ and $(g^{\frac{n}{p}})^{b_0}$. This can be done efficiently using the repeated squaring trick.

Inductively, suppose we've computed b_{i-1} such that $x \equiv b_{i-1} \pmod{p^i}$. Now we want to compute j such that $x \equiv b_{i-1} + jp^i \pmod{p^{i+1}}$ for $0 \leq j \leq p-1$.

$$\begin{aligned} x &\equiv b_{i-1} + jp^i \pmod{p^{i+1}} \\ \Leftrightarrow g^x &= g^{b_{i-1} + jp^i} \\ \Leftrightarrow y &= g^{b_{i-1} + jp^i} \\ \Leftrightarrow y^{\frac{n}{p^{i+1}}} &= g^{\frac{n}{p^{i+1}}(b_{i-1} + jp^i)} \\ \Leftrightarrow y^{\frac{n}{p^{i+1}}} &= g^{\frac{n}{p^{i+1}}b_{i-1}} g^{\frac{n}{p}j} \\ \Leftrightarrow y^{\frac{n}{p^{i+1}}} (g^{\frac{n}{p^{i+1}}b_{i-1}})^{-1} &= g^{\frac{n}{p}j} \quad (*) \end{aligned}$$

We try $j = 0, 1, \dots, p-1$ and when there is equality between $y^{\frac{n}{p^{i+1}}} (g^{\frac{n}{p^{i+1}}b_{i-1}})^{-1}$ and $g^{\frac{n}{p}j}$ we get our $b_i \leftarrow b_{i-1} + jp^i$.

Observe that the LHS of (*) is in $\langle g^{\frac{n}{p}} \rangle$ since $(y^{\frac{n}{p^{i+1}}})^p = (y^{\frac{n}{p}})^p = g^{\frac{n}{p}b_{i-1}}$.

For each prime p , each b_i requires p trials and there are $\alpha \sim \log n$ many b_i 's to be computed. Now this computation is done for the atmost $\log n$ many

primes dividing n , so we need $O(p \log^2 n)$ group operations. The primes are bounded by k and therefore the time complexity is $O(k \log^2 n)$ group ops. If n is a prime then it is n -smooth, so $k = n$ and the algorithm takes $O(n \log^2 n)$ which is exponential in $\log n$. On the other hand if n is $\log n$ -smooth that is $k = \log n$ then we are in polynomial time in $\log n$.

2. Index Calculus attack

The reason this algorithm is called the Index Calculus method probably because the discrete logarithm of an element is also called it's index. This technique solves the DL problem over \mathbf{F}_p in time subexponential in $\log p$. The underlying idea is the same as in the \mathbf{F}_{p^n} case which Koblitz presents in the textbook. This attack relies on the *smoothness trick*. Next week we'll see integer factoring algorithms which make use of the smoothness trick.

The DL problem in this setting is as follows. Say $\mathbf{F}_p^* = \langle g \rangle$. Given $y \in \mathbf{F}_p^*$ to compute x such that $y = g^x$.

Smoothness trick. Suppose we are working with $\mathbf{F}_{131}^* = \langle g \rangle$ and want to compute the discrete logarithm of **100** base g . But viewing **100** $\in \mathbf{F}_{131}^*$ instead as our friendly number $100 \in \mathbf{Z}$ we see that $100 = 5^2 2^2$ factors *smoothly* into small primes.

Now if somebody gave us α and β the discrete logarithms of **5** and **2** base g respectively, that is $\mathbf{5} = g^\alpha$ ($5 \equiv g^\alpha \pmod{131}$) and $\mathbf{2} = g^\beta$ ($2 \equiv g^\beta \pmod{131}$) then we can calculate the discrete logarithm of **100** base g realizing that $100 \equiv (g^\alpha)^2 (g^\beta)^2 \pmod{131} \equiv g^{2\alpha+2\beta} \pmod{131}$. And since the order of the group is 130, the discrete logarithm of **100** base g is $2\alpha + 2\beta \pmod{130}$.

So the smoothness trick is to impose an integer structure on the field \mathbf{F}_p in other words to view ("lift") the elements of \mathbf{F}_p as elements in \mathbf{Z} .

To apply this trick we first set a smoothness bound b . The strategy is to randomize y so that it becomes b -smooth as follows. Pick a random a and compute $[g^a y]$ the reduced representative of $g^a y$ which is equal to $g^a y \pmod p$. As a random $\# \times$ a fixed $\#$ is a random $\#$, $g^a y \pmod p$ is a random element of the $\{0, 1, \dots, p-1\}$ since a is random.

Now with a little help from our friend $\#$ Theory. In what follows n doesn't have to be prime.

Definition.

$$\psi(n, m) = \#\{x \leq n \mid x \text{ is } m\text{-smooth}\}$$

Notation.

$$L_n(\alpha, c) = e^{c(\log n)^\alpha (\log \log n)^{1-\alpha}}$$

For fixed n and α , $L_n(\alpha, c)$ is a polynomially related class for the various c 's. $L_n(0, c) = (\log n)^c$ is the polynomial-in- $\log n$ class. At the other extreme is $L_n(1, c) = n^c$ the exponential-in- $\log n$ class. When $0 < \alpha < 1$, we have the subexponential classes.

$L_n(\alpha, c) = (L_n(\alpha, 1))^c$ and $L_n(\alpha, 2c) = (L_n(\alpha, c))^2$ and so in practice c plays an important role and when talking about time estimates the smaller the c the better. But for the purposes of this discussion we ignore the constant c and introduce the following modified notation.

Notation.

$$L_n[\alpha] = e^{O((\log n)^\alpha (\log \log n)^{1-\alpha})}$$

Fact. If $\alpha \leq \beta \leq 1$ then

$$\frac{\psi(L_n[\beta], L_n[\alpha])}{L_n[\beta]} \geq \frac{1}{L_n[\beta - \alpha]}$$

Taking $\beta = 1$, $L_n[\beta] = L_n[1] = n^{O(1)}$ we get

$$\frac{\psi(n, L_n[\alpha])}{n} \geq \frac{1}{L_n[1 - \alpha]}$$

So if we take α to be the “optimal” value $\frac{1}{2}$, n to be p the size of our finite field and set smoothness bound b to be $L_p[\frac{1}{2}] = e^{O(\sqrt{\log p} \log \log p)}$ then

$$\frac{\psi(p, L_p[\frac{1}{2}])}{p} \geq \frac{1}{L_p[\frac{1}{2}]}$$

So b has roughly $\sqrt{\log p} \log \log p$ bits and ignoring the $\log \log p$ part it has $\sqrt{\log p}$ bits. Observe that $\sqrt{\log p}$ is way smaller than $\log p$. If $\log p \sim 1024$ then $\sqrt{\log p} \sim 36$. So we are going to collect all primes $p_i \leq b$ and call them our “(factor) base primes”.

For random a random number a , $g^a y \bmod p$ is b -smooth with probability atleast b^{-c} (where c is a constant which can be calculated) and so in b^c many

trials of random a 's we expect to get a b -smooth number. Now suppose we hit up on such a $g^a y \bmod p$ number which is b -smooth, so $(g^a y \bmod p) = \prod_{p_i \leq b} p_i^{e_i}$ that is $(g^a y \bmod p) \equiv \prod_{p_i \leq b} p_i^{e_i} \bmod p$. Note that to factor a b -smooth number takes time polynomial in b .

Also suppose $p_i = g^{x_i}$ and since $y = g^x$ we get $g^{a+x} = g^a g^x \equiv g^{\sum x_i e_i} \bmod p \Rightarrow a + x \equiv \sum x_i e_i \bmod p - 1$.

So each b -smooth $g^a y \bmod p$ gives rise to a linear congruence in the unknowns x and x_i and the number of unknowns is atmost b . Hence the number of equations we need to compute x is atleast b . Now we iterate and come up b many b -smooth $g^a y \bmod p$ numbers to we end up with a linear system with b many modular congruence relations which look like: $a + x \equiv \sum x_i e_i \bmod p - 1$.

A catch is that we are dealing with modular congruences so our standard linear algebra techniques will not help. But as the associated matrix (with the rank of the matrix which is atmost b) is quite sparse there is an efficient algorithm polynomial in the rank of the matrix. Now all the work done is polynomial in b and so the running time is $b^{O(1)} = e^{O(\sqrt{\log p \log \log p})}$ which is a subexponential in b .

Once we solve the system as a bonus we also get all the x_i 's the discrete logarithms of the primes in our factor base. So for another instance of the DL problem $y' = g^{x'}$ all we need is just need one more congruence relation. So generating the x_i 's can be thought of as a pre-computation stage.

We've succeeded in developing a $L[\frac{1}{2}]$ algorithm, can we push it further? Yes by observing that the probability for an $L[\frac{2}{3}]$ number to be $L[\frac{1}{3}]$ -smooth is $\frac{1}{L[\frac{2}{3}-\frac{1}{3}]} = \frac{1}{L[\frac{1}{3}]}$ and doing some more fancy things.

So the success of this attack relies on lifting finite prime field elements to the integers. What about \mathbf{F}_{p^n} ? Can we play a similar kind of trick. Note that this is not the set of numbers between 1 and $p^n - 1$ but $\mathbf{F}_{p^n} = \mathbf{F}_p[x]/h = \{a_0 + a_1 x + \dots + a_{n-1} x^{n-1} \mid a_i \in \mathbf{F}_p\}$ where h is an irreducible polynomial of degree n in $\mathbf{F}_p[x]$ and say $\mathbf{F}_{p^n} = \langle g \rangle$. DL problem is to calculate e such that $y \equiv g^e \bmod h$. In this setting polynomials decompose into irreducible polynomials and we set a smoothness bound on the degree $b \sim \sqrt{n} \log p$ or $b \sim \sqrt{n}$ for fixed p . And we get linear congruences as follows:

$g^a \bmod h = \prod_{\deg(h_i) \leq b} h_i^{a_i} \Rightarrow a \equiv \sum a_i x_i \bmod p^n - 1$ (since the order of the group is $p^n - 1$). And this is the DL over \mathbf{F}_{p^n} algorithm described in the book [K] § IV.3.

3. Birthday Paradox Attack

Birthday Paradox: How many people must be there in a room so that atleast 2 were born on the same day of the year with probability $\geq \frac{1}{2}$?

Answer: 28 and not 366!

Proof: Say that a year has n days and there are k persons in the room whose birthdays are represented by $x_i, 1 \leq i \leq k$ and the x_i 's are independent uniformly distributed random variables which take values from the set of birthdays $B = \{1, 2, \dots, n\}$. *Uniformly distributed* means $\Pr[x_i = a] = \frac{1}{n}$ for all $a \in B$ and for all r.v. x_i . *Independent* means $\Pr[x_i = a \text{ and } x_j = b] = \Pr[x_i = a] \times \Pr[x_j = b] = \frac{1}{n^2}$ for all $a, b \in B$ and for all r.v. $x_i, x_j, i \neq j$.

Let's define X_{ij} to be a random (indicator) variable and

$$X_{ij} = \begin{cases} 1, & \text{if } x_i = x_j \text{ that is they have the same birthday} \\ 0, & \text{otherwise} \end{cases}$$

Now let us calculate it's expectation

$$\begin{aligned} E(X_{ij}) &= 1 \cdot \Pr[X_{ij} = 1] + 0 \cdot \Pr[X_{ij} = 0] = \sum_{m=1}^n \Pr[x_i = x_j = m] \\ &= \sum_{m=1}^n \frac{1}{n^2} = \frac{n}{n^2} = \frac{1}{n} \end{aligned}$$

Let's define another random variable $X = \sum_{i \neq j} X_{ij}$ and

$$\begin{aligned} E(X) &= E(\sum_{i \neq j} X_{ij}) = \sum_{i \neq j} E(X_{ij}) \text{ due to linearity of expectation} \\ &= \binom{k}{2} \frac{1}{n} \end{aligned}$$

We would like $E(X) \geq 1$ and for this it is sufficient that $k \geq \frac{\sqrt{8n+1}}{2}$. The minimum k when we plug 365 for n turns out to be 28.

So if you have a *random enough* function f on a set of cardinality n , then the size of the subset with atleast 2 in the subset having the same image under f with *good* probability is \sqrt{n}

The Birthday Paradox attack is a general purpose technique which can be used to compute the discrete log in a cyclic group G in exponential time $\sim O(\sqrt{|G|})$ group operations (which is better than the $O(|G|)$ naive brute-force algorithm). The crux of the algorithm is the Birthday Paradox and hence the name, it is as called the Pollard-Rho attack.

DL problem. Given $y \in G = \langle g \rangle$ to compute x such that $y = g^x$.

Consider the following bijective map f ,

$$f : G \rightarrow G$$

$$z \mapsto z^a g^b$$

where a, b are random integers and $(a, n) = 1$, where $n = |G|$.

Also consider the following iterative way to generate new elements in G starting with the element y :

$$y_1 = f(y) = y^a g^b = g^{ax} g^b = g^{ax+b}$$

$$= g^{a_1 x + b_1}, \text{ where } a_1 = a, b_1 = b$$

$$y_2 = f(y_1) = y_1^a g^b = g^{a(ax+b)} g^b$$

$$= g^{a^2 x + ab + b} = g^{a_2 x + b_2}, \text{ where } a_2 = a^2, b_2 = ab + b$$

$$\dots$$

$$y_i = f(y_{i-1}) = g^{a_i x + b_i}, \text{ where } a_i = a^i, b_i = \sum_{j=0}^{i-1} a^j b$$

$$\dots$$

Since f is a random function, we can think of y_1, y_2, \dots as a random sequence of elements in G . By the *Birthday problem* paradox we'll encounter $y_i = y_j$, with $i \neq j$ in about $O(\sqrt{n})$ iterations with probability ≥ 0.5 .

$$y_i = y_j, i \neq j$$

$$\Rightarrow g^{a_i x + b_i} = g^{a_j x + b_j}$$

$$\Rightarrow a_i x + b_i \equiv a_j x + b_j \pmod{n}$$

$$\Rightarrow x = (b_j - b_i)(a_i - a_j)^{-1} \pmod{n}$$

Therefore we start with $y = y_1$ and generate y_i 's until we compute an element we have already generated y_j and as we know a_i, b_i, a_j, b_j we can recover x . Hence instead of enumerating all possibilities - the naive method we essentially do a random walk (which looks the symbol rho: ρ , hence the name for this algorithm). We can also parallelize this scheme to get better results. Pollard-rho is one kind of random walk and any other similar scheme will also work.