Joux's Recent Index Calculus Results

Part I: Introduction to the Computational Discrete Logarithm Problem

Joshua E. Hill

Department of Mathematics, University of California, Irvine

Cryptography Seminar
May 14, 2013
`http://bit.ly/129p1If`

v1.3

Talk Outline

Introduction Outline

Subsection 1

The Discrete Log Problem

Discrete and Discreet

Definition

Given a finite group G (written multiplicatively), and a generator $g \in G$, given $t = g^\ell$ for some $\ell \in \mathbb{Z}$, calculate ℓ. This is called the discrete logarithm, and is denoted $\log_g(t) = \ell$.

- The difficulty of performing discrete logs is the foundational hardness assumption for much of cryptography (*e.g.*, Diffie-Hellman and its variants, El Gamal and its variants).

Running With the Wrong Group

- The difficulty of this problem is profoundly dependent on the underlying group.
- All finite cyclic groups are (group-)isomorphic to $\mathbb{Z}/n\mathbb{Z}$ (under addition), where $n = |g|$.
- If we can use the (group-)isomorphism induced by $g \mapsto 1$, the problem becomes trivial.

Jets *vs.* Sharks

Example (Multiplicative)

$(\mathbb{Z}/107\mathbb{Z})^\times$ is a cyclic group of order 106, which has subgroups of order $1, 2, 53, 106$. The element $g = 3$ clearly doesn't have order 1 or 2, but $3^{53} \equiv 1 \pmod{107}$, so g generates a group of order 53. It is not clear how to efficiently calculate $\log_3(19)$.

$\updownarrow$

Example (Additive)

$\mathbb{Z}/106\mathbb{Z}$ is a cyclic group of order 106. The element $g = 2$ clearly has order 53. The corresponding problem is $\log_2(84) = 42$.

Solving the discrete log problem for every value in the cyclic subgroup completely describes this isomorphism.

Subsection 2

Time Complexity Notes

Big-O Notation (and Family)

- We have two eventually positive real valued functions $A, B : \mathbb{N}^k \to \mathbb{R}$. Take $\mathbf{x}$ as an n-tuple, with $\mathbf{x} = (x_1, \ldots, x_n)$
- We'll write $|\mathbf{x}|_{\text{min}} = \min_i x_i$.

Definition

$A(\mathbf{x}) = O(B(\mathbf{x}))$ if there exists a positive real constant C and an integer N so that if $|\mathbf{x}|_{\text{min}} > N$ then $A(\mathbf{x}) \leq CB(\mathbf{x})$. (*i.e.* A is bounded above by B asymptotically.)

Definition

$A(\mathbf{x}) = o(B(\mathbf{x}))$ if for all positive real constants C there is an integer N so that if $|\mathbf{x}|_{\text{min}} > N$ then $A(\mathbf{x}) \leq CB(\mathbf{x})$. (*i.e.* A is dominated by B asymptotically.)

"When I Use a Word..."

Definition

An algorithm is considered polynomial time if it is time complexity $O(x^k)$ where k is a fixed positive integer and x is the input length.

Definition

An algorithm is considered exponential time if it is time complexity $O(2^{x^k})$ where k is a fixed positive integer, and x is the input length.

Definition

An algorithm is considered sub-exponential time if it is time complexity $2^{o(x)}$ where x is the input length.

"It Means Just What I Choose it to Mean"

- These definitions are dependent on the group and its representation.
- Note that a group of order q takes (on average) at least $\lceil \log_2(q) \rceil$ bits to represent a group element.
- The log function thus takes an input of $2 \lceil \log_2(q) \rceil$ bits.
- For the purposes of this discussion, imagine $n \sim q$.

Introduction Outline

1 Introduction

2 Classical Approaches to the Discrete Log Problem
- Reductions
- Exponential Computational Approaches
- Subexponential Computational Approaches

3 Conclusion, Mk. I

Introduction Outline

(n.b., these are “Classical” in the graduate student sense.)

Subsection 1

Reductions

First Steps

- The difficulty of performing the discrete log operation is determined by the size of the cyclic group $|g| = \#(\langle g \rangle) = n$ and the group in which it is embedded.
- If n is composite (and can be factored), various reductions are possible.
- These are collectively often known as "The Pohlig–Hellman Algorithm".

Composite Reduction #1

If $n = uv$ and $\gcd(u, v) = 1$, we can solve the discrete log by solving separate discrete logs in a u and v ordered group.

- There are integers a, b so that $au + bv = 1$.
- $|g^u| = v$ and $|g^v| = u$.
- If we knew $\log_{g^u}(t^u) = \ell_u$ and $\log_{g^v}(t^v) = \ell_v$, then

$$t = t^{au+bv} = g^{u\ell_u a} g^{v\ell_v b} = g^{u\ell_u a + v\ell_v b}$$

Composite Reduction #2

If $n = p^a$ then we can solve by performing a logs in an order p group.

- We seek to calculate $\ell = \log_g(t)$.
- We can represent ℓ base p as $\ell = \sum_{j=0}^{a-1} b_j p^j$ (where $0 \le b_i < p$)
- We can solve for each b_j in succession by performing a calculation in a group of order p:
  - We can solve for b_0 as $t^{p^{a-1}} = g^{\ell p^{a-1}} = g^{b_0 p^{a-1}} = \left(g^{p^{a-1}}\right)^{b_0}$.
  - To solve for b_j, if we know b_0 to b_{j-1}, and let $t_j = tg^{-b_0 - b_1 p - \ldots - b_{j-1} p^{j-1}}$, so then t_j^{a-j-1} is in $\left\langle g^{p^{a-1}} \right\rangle$, so we can solve $\log_{g^{p^{a-1}}}\left(t_j^{p^{a-j-1}}\right)$.

Reduction Summary

- We can reduce problems of finding logarithms on groups with composite order to (possibly much easier) subproblems.
- This isn't desirable for cryptography, so almost all cryptographic settings require that g generates either:
  - a very large prime ordered group, or
  - a very large group whose order is impractical to factor.
- We can restrict our discussion to groups of prime order.

Subsection 2

Exponential Computational Approaches

Pick Your Poison: Exhaustion

Deterministic Approaches

- Brute force, requires on average $n/2$ group operations (time complexity is $O(n)$ group operations), negligible storage.

Example

In $(\mathbb{Z}/107\mathbb{Z})^\times$, calculate $\log_3(19)$.

j	0	1	2	$\cdots$
3^j				

Pick Your Poison: Exhaustion

Deterministic Approaches

- Brute force, requires on average $n/2$ group operations (time complexity is $O(n)$ group operations), negligible storage.

Example

In $(\mathbb{Z}/107\mathbb{Z})^{\times}$, calculate $\log_3(19)$.

j	0	1	2	$\cdots$
3^j	1			

Pick Your Poison: Exhaustion

Deterministic Approaches

- Brute force, requires on average $n/2$ group operations (time complexity is $O(n)$ group operations), negligible storage.

Example

In $(\mathbb{Z}/107\mathbb{Z})^\times$, calculate $\log_3(19)$.

j	0	1	2	$\cdots$
3^j	1	3		

Pick Your Poison: Exhaustion

Deterministic Approaches

- Brute force, requires on average $n/2$ group operations (time complexity is $O(n)$ group operations), negligible storage.

Example

In $(\mathbb{Z}/107\mathbb{Z})^{\times}$, calculate $\log_3(19)$.

j	0	1	2	$\cdots$
3^j	1	3	9	

Pick Your Poison: Exhaustion

Deterministic Approaches

- Brute force, requires on average $n/2$ group operations (time complexity is $O(n)$ group operations), negligible storage.

Example

In $(\mathbb{Z}/107\mathbb{Z})^\times$, calculate $\log_3(19)$.

j	0	1	2	$\cdots$
3^j	1	3	9	$\cdots$

Pick Your Poison: Exhaustion

Deterministic Approaches

- Brute force, requires on average $n/2$ group operations (time complexity is $O(n)$ group operations), negligible storage.

Example

In $(\mathbb{Z}/107\mathbb{Z})^{\times}$, calculate $\log_3(19)$.

j	0	1	2	$\cdots$	41
3^j	1	3	9	$\cdots$	42

Deterministic Approaches

- Brute force, requires on average $n/2$ group operations (time complexity is $O(n)$ group operations), negligible storage.

Example

In $(\mathbb{Z}/107\mathbb{Z})^\times$, calculate $\log_3(19)$.

j	0	1	2	$\cdots$	41	42
3^j	1	3	9	$\cdots$	42	19

Pick Your Poison: Drowning

- The Baby Steps, Giant Steps algorithm [Shanks, 1971]
  - We seek to calculate $\ell = \log_g(t)$.
  - If we let $m = \lceil \sqrt{n} \rceil$, we can write ℓ in base m as $\ell = b_0 + b_1 m$ (with $0 \leq b_i < m$) . We then see $g^{\ell} = g^{b_0 + b_1 m} = t$, so $g^{-b_1 m} t = g^{b_0}$.
  - Calculate the list $\{g^0, \ldots, g^{m-1}\}$, add them to a hash table, and then step through the (at most) m calculations ($j \in \{0, 1, \ldots, m-1\}$) for $g^{-jm} t$ until a collision is found. $O(\sqrt{n})$ group operations and storage.

First, Find a Nice Lake

Example

In $(\mathbb{Z}/107\mathbb{Z})^\times$, calculate $\log_3(19)$; we thus have $m = \left\lceil \sqrt{53} \right\rceil = 8$. First, calculate the Baby Steps:

j	0	1	2	3	4	5	6	7
3^j	1	3	9	27	81	29	87	47

Now calculate the Giant Steps:

j	0	1	2	3	4	5	6	7
$3^{-8j} \cdot 19$								

First, Find a Nice Lake

Example

In $(\mathbb{Z}/107\mathbb{Z})^\times$, calculate $\log_3(19)$; we thus have $m = \left\lceil \sqrt{53} \right\rceil = 8$. First, calculate the Baby Steps:

j	0	1	2	3	4	5	6	7
3^j	1	3	9	27	81	29	87	47

Now calculate the Giant Steps:

j	0	1	2	3	4	5	6	7
$3^{-8j} \cdot 19$	19							

First, Find a Nice Lake

Example

In $(\mathbb{Z}/107\mathbb{Z})^\times$, calculate $\log_3(19)$; we thus have $m = \left\lceil \sqrt{53} \right\rceil = 8$. First, calculate the Baby Steps:

j	0	1	2	3	4	5	6	7
3^j	1	3	9	27	81	29	87	47

Now calculate the Giant Steps:

j	0	1	2	3	4	5	6	7
$3^{-8j} \cdot 19$	19	10						

First, Find a Nice Lake

Example

In $(\mathbb{Z}/107\mathbb{Z})^\times$, calculate $\log_3(19)$; we thus have $m = \left\lceil \sqrt{53} \right\rceil = 8$. First, calculate the Baby Steps:

j	0	1	2	3	4	5	6	7
3^j	1	3	9	27	81	29	87	47

Now calculate the Giant Steps:

j	0	1	2	3	4	5	6	7
$3^{-8j} \cdot 19$	19	10	101					

First, Find a Nice Lake

Example

In $(\mathbb{Z}/107\mathbb{Z})^{\times}$, calculate $\log_3(19)$; we thus have $m = \left\lceil \sqrt{53} \right\rceil = 8$. First, calculate the Baby Steps:

j	0	1	2	3	4	5	6	7
3^j	1	3	9	27	81	29	87	47

Now calculate the Giant Steps:

j	0	1	2	3	4	5	6	7
$3^{-8j} \cdot 19$	19	10	101	25				

First, Find a Nice Lake

Example

In $(\mathbb{Z}/107\mathbb{Z})^\times$, calculate $\log_3(19)$; we thus have $m = \left\lceil \sqrt{53} \right\rceil = 8$. First, calculate the Baby Steps:

j	0	1	2	3	4	5	6	7
3^j	1	3	9	27	81	29	87	47

Now calculate the Giant Steps:

j	0	1	2	3	4	5	6	7
$3^{-8j} \cdot 19$	19	10	101	25	92			

First, Find a Nice Lake

Example

In $(\mathbb{Z}/107\mathbb{Z})^{\times}$, calculate $\log_3(19)$; we thus have $m = \left\lceil \sqrt{53} \right\rceil = 8$. First, calculate the Baby Steps:

j	0	1	2	3	4	5	6	7
3^j	1	3	9	27	81	29	87	47

Now calculate the Giant Steps:

j	0	1	2	3	4	5	6	7
$3^{-8j} \cdot 19$	19	10	101	25	92	9		

First, Find a Nice Lake

Example

In $(\mathbb{Z}/107\mathbb{Z})^{\times}$, calculate $\log_3(19)$; we thus have $m = \lceil \sqrt{53} \rceil = 8$. First, calculate the Baby Steps:

j	0	1	2	3	4	5	6	7
3^j	1	3	9	27	81	29	87	47

Now calculate the Giant Steps:

j	0	1	2	3	4	5	6	7
$3^{-8j} \cdot 19$	19	10	101	25	92	9		

$\ell = 5 \cdot 8 + 2 = 42$

"Your Friend Here is Only *Mostly* Dead!"

Probabilistic Approaches:

- Pollard's ρ-method [Pollard 1978] to calculate $\ell = \log_g(t)$.
  - Attempt to find an $a_i, \hat{a}_i, b_i, \hat{b}_i \in \mathbb{Z}/n\mathbb{Z}$ so that $g^{a_i} t^{b_i} = g^{\hat{a}_i} t^{\hat{b}_i}$.
  - ℓ is then a solution to $\left(\hat{b}_i - b_i\right) \ell \equiv (a_i - \hat{a}_i) \pmod{n}$.
  - Pseudo-randomly explore the group using an iterated function $x_i = f(x_{i-1})$ defined so that $x_i = g^{a_i} t^{b_i}$; start at $x_0 = 1 = g^0 t^0$.
  - We hope to encounter a cycle.
  - Cycle detection using Floyd's cycle detection algorithm.

"I'm as Mad as Hell..."

- Some notes on Pollard's ρ-method:
  - Encountering a cycle with a random map is expected to occur after $\sqrt{\frac{\pi n}{2}}$ steps.
  - We heuristically assume that our function has this same behavior.
  - The expected time complexity is then $O\left(\sqrt{n}\right)$ group operations, with negligible storage required.
  - This can fail in the (unlikely) event that the collision is due to $a_i \equiv \hat{a}_i \pmod{n}$ (and thus $b_i \equiv \hat{b}_i \pmod{n}$).
  - This can be abstracted to a parallelizable algorithm by starting each process at a random index a_i and watching for collisions between processes (Pollard's λ-method).

“And I’m Not Going to Take This Anymore!”

- Partition $\langle g \rangle$ into three sets of roughly the same size, S_0, S_1, S_2.
- Define

$$f(x) = \begin{cases} tx & x \in S_0 \\ x^2 & x \in S_1 \\ gx & x \in S_2 \end{cases}$$

- If we step through as $x_{i+1} = f(x_i)$ with $x_i = g^{a_i} t^{b_i}$, then we can examine the exponents:

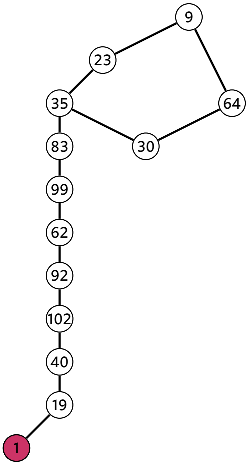
$$f(g^{a_i} t^{b_i}) = \begin{cases} g^{a_i} t^{b_i+1} & g^{a_i} t^{b_i} \in S_0 \\ g^{2a_i} t^{2b_i} & g^{a_i} t^{b_i} \in S_1 \\ g^{a_i+1} t^{b_i} & g^{a_i} t^{b_i} \in S_2 \end{cases}$$

Like the Chase Scene in "The French Connection"

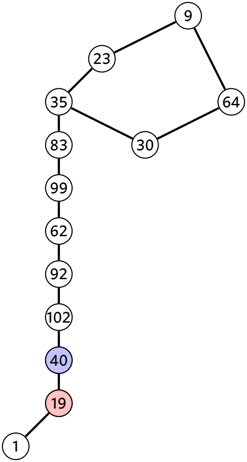
Example

In $(\mathbb{Z}/107\mathbb{Z})^\times$, calculate $\log_3(19)$. Let $S_0 = \{0, 1, \ldots 35\}$, $S_1 = \{36, 37, \ldots 71\}$, and $S_2 = \{72, 73, \ldots 106\}$.

ρ

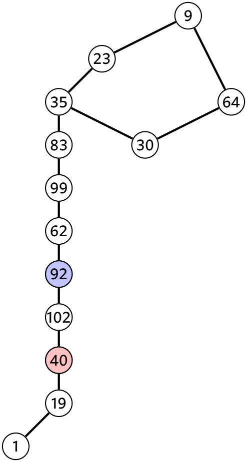


i	a_i	b_i	x_i	a_{2i}	b_{2i}	x_{2i}
0	0	0	1	0	0	1



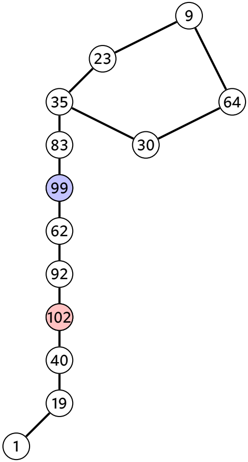
i	a_i	b_i	x_i	a_{2i}	b_{2i}	x_{2i}
1	0	1	19	0	2	40

ρ



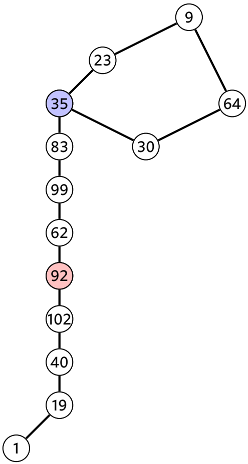
i	a_i	b_i	x_i	a_{2i}	b_{2i}	x_{2i}
2	0	2	40	1	4	92

ρ



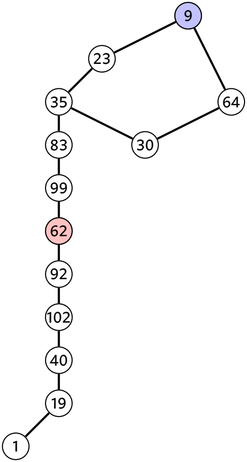
i	a_i	b_i	x_i	a_{2i}	b_{2i}	x_{2i}
3	0	4	102	4	8	99

ρ, ρ



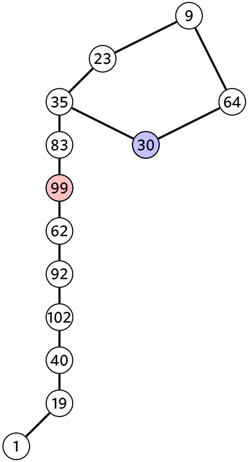
i	a_i	b_i	x_i	a_{2i}	b_{2i}	x_{2i}
4	1	4	92	6	8	35

ρ, ρ



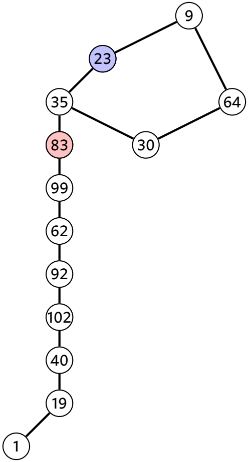
i	a_i	b_i	x_i	a_{2i}	b_{2i}	x_{2i}
5	2	4	62	6	10	9

ρ, ρ



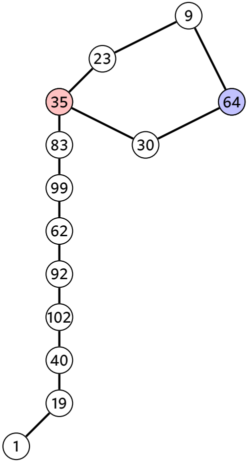
i	a_i	b_i	x_i	a_{2i}	b_{2i}	x_{2i}
6	4	8	99	12	22	30

ρ, ρ



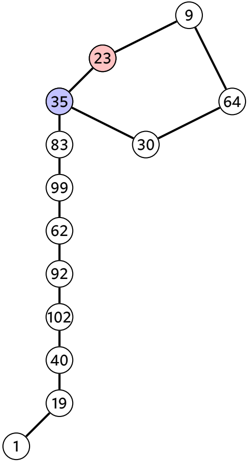
i	a_i	b_i	x_i	a_{2i}	b_{2i}	x_{2i}
7	5	8	83	12	24	23

ρ, ρ, ρ



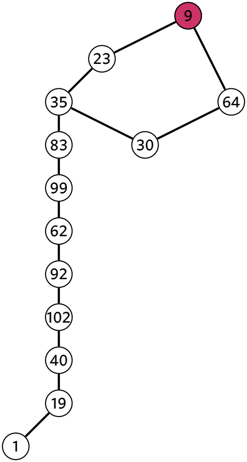
i	a_i	b_i	x_i	a_{2i}	b_{2i}	x_{2i}
8	6	8	35	12	26	64

ρ, ρ, ρ your



i	a_i	b_i	x_i	a_{2i}	b_{2i}	x_{2i}
9	6	9	23	24	53	35

ρ, ρ, ρ your boat...



i	a_i	b_i	x_i	a_{2i}	b_{2i}	x_{2i}
10	6	10	9	24	55	9

A Photo Finish!

This gives us:

$$\ell \equiv (55 - 10)^{-1}(6 - 24) \equiv 42 \pmod{53}$$

A Note on $O(\sqrt{n})$

- $n = e^{\log n}$ so $O(\sqrt{n}) = O\left(e^{\frac{1}{2}\log n}\right)$ is clearly exponential in the size of n.
- Our assumption that $n \sim q$ gives us that this is exponential in the input size.
- All of these computations are with respect to group operations. The time complexity of performing these operations is highly dependent on the group, and varies with q (generally, polynomial in the size of q).

Subsection 3

Subexponential Computational Approaches

I don’t like this problem. Let’s change it!

- We first look at the Index Calculus Method [Kraitchik, 1922] and [Hellman-Reyneri, 1983] .
- We break down the problem into subproblems.
- If we could represent our group as the unit group in a homomorphic image of $\mathbb{Z}$, then we can leverage some structure from $\mathbb{Z}$.
- We will seek relations between factorizations of numbers of the form $g^r \pmod{n}$.

That's SMOOTH!

- If we want relations based on primes found in random integers, we want to pay attention to the primes that occur most often.
- For randomly selected integers within our bounds, small primes will occur as factors of these random numbers more often than large primes.
- We are thus interested in small primes, hence smooth integers.

Definition

An integer is B-smooth if its factorization involves only primes less than or equal to B.

All Your Factor Base are Belong to Us.

- Let's assume we are working in $G = (\mathbb{Z}/p\mathbb{Z})^\times$ for some odd prime p, and $\langle g \rangle = G$.
- Establish the smoothness bound $B \ll p$.
- Refer to the k primes less than or equal to B as $p_1, \ldots, p_k$. These are called the factor base. By convention, we let $p_0 = -1$.
- Generate g^r where r is chosen randomly in $[0, p-1]$.
- Factor g^r. If it is B-smooth, then we have found a relation, namely $g^r = \prod_{i=0}^{k} p_i^{e_i}$. This corresponds to the additive relation

$$r \equiv \sum_{i=0}^{k} e_i \log_g(p_i) \pmod{p-1}$$

Take Off Every 'Zig'!

- We already know that $\log_g(-1) = \frac{p-1}{2}$.
- If we collect k independent relations, then we can use linear algebra to solve for the values of each of the $\log_g(p_i)$'s.
- We now have a way of finding the logarithm of any B-smooth integer; if t is B-smooth, then

$$\log_g(t) \equiv \sum_{i=0}^{k} e_i \log_g(p_i) \pmod{p-1}.$$

- If t is not B-smooth, we could try to find the logarithm of a related value...
  - We randomly search for r so that tg^r is B-smooth.
  - Once we find such an r, we then have:

$$\log_g(t) \equiv -r + \sum_{i=0}^{k} e_i \log_g(p_i) \pmod{p-1}$$

Hold on There, Sparky!

- "You just assumed that we could factor integers that look like $g^r \pmod{p}$. They could be… large!"
  - We could just use trial division, as we only care about a particular small set of primes.
  - Lenstra's Elliptic Curve Factoring method is a polynomial time method for a sufficiently dense set of smooth integers.
- "You just assumed that we could do linear algebra mod $(p-1)$, which I think implies $p = 3$!"
  - There are a few options to overcome this problem.
  - You could couple Hensel-style lifting, and then combine results using the CRT.
  - It may also "just work" if you don't need to invert anything that is a factor of $(p-1)$.
  - You could also choose your relations specifically so this step "just works".

"Do the Bomb Bay Door Thing."

As a note, for consistency, we are operating in a subgroup of index 2 here, so some behavior changes.

Example

In $(\mathbb{Z}/107\mathbb{Z})^\times$, calculate $\log_3(19)$. We set $B = 13$, so our factor base is $\{3, 11, 13\}$ (we have discarded $\{-1, 2, 5, 7\}$ as they are not in $\langle g \rangle$). We randomly choose several r, searching for values of g^r that can be expressed using our factor base:

r	$3^r \pmod{107}$	p_1 3	p_2 11	p_3 13
3	27	3	0	0
22	99	2	1	0
33	39	1	0	1

"Bomb Bay Doors Swinging and Open, Baby!"

Example

This corresponds to the additive relations:

$$3 \equiv 3\log_3(3) \pmod{53} \quad (1)$$

$$22 \equiv 2\log_3(3) + 1\log_3(11) \pmod{53} \quad (2)$$

$$33 \equiv 1\log_3(3) + 1\log_3(13) \pmod{53} \quad (3)$$

Equation (1) clearly gives $\log_3(3) = 1$. Equations (2) and (1) give $\log_3(11) = 20$, and equations (3) and (1) give $\log_3(13) = 32$.

"Groovy and out."

Example

- 19 is clearly not 13-smooth.
- Now randomly select r, looking for a 13-smooth $19 \cdot 3^r$.
- We find $19 \cdot 3^{44} \equiv 39 = 3 \cdot 13 \pmod{107}$, thus

$$\log_3(19) \equiv \log_3(3) + \log_3(13) - 44 \pmod{53}$$
$$\log_3(19) \equiv 1 + 32 - 44 \equiv 42 \pmod{53}$$

Parameters and Performance

- Choosing an optimal B is complicated. See [Poonen 2008].
- This class of algorithms share a time complexity class, namely $L_n(\alpha, c)$ where

$$L_n(\alpha, c) = \exp\left((c + o(1)) (\log n)^{\alpha} (\log \log n)^{1-\alpha}\right)$$

- $L_n(\alpha, c)$ is sub-exponential.
- Using Lenstra's elliptic curve factoring method to factor candidates has time complexity in $L_p\left(1/2, \sqrt{2}\right)$ for optimal choice of B.
- Additional work to solve for the $\log_g t$ is $L_p\left(1/2, 1/\sqrt{2}\right)$. If a larger than needed B was selected, this step is even faster.

To Infinity… *and Beyond!*

- The fundamental notion that must be abstracted to apply this algorithm to other groups is the notion of *smoothness*.
- In some cases, this abstracts clearly, and the algorithm directly applies.
  - In $\mathbb{F}_{p^a}$, examine the representation of elements as $\mathbb{F}_p[x]/(f(x))$ where $f(x)$ is a degree a irreducible polynomial.
  - $\mathbb{F}_p[x]$ is a UFD, so we can define smoothness with respect to the degree of the irreducibles in the factorization of a canonical polynomial in $\mathbb{F}_p[x]$ used to represent the element.
  - B-smooth in this context means that no irreducible factor has degree greater than B.
  - This notion of smoothness directly yields a sub-exponential algorithm for computing the discrete logarithm problem. [Bender-Pomerance, 1998]

Good News Everyone!

- Not all groups currently have a notion of *smoothness*.
  - Elliptic curves have no analogous notion at present, which means that this technique (class) doesn't apply.
  - This is one reason that the generalized elliptic curve discrete log problem is still exponential.
  - With elliptic curves with low embedding degree we can proceed by reducing the ECDLP to a DLP over a finite field.

Is This Talk *EVER* Going to End?

- We can further abstract by applying a set of index calculus techniques derived from the Number Field Sieve.
- This leads to a few related approaches:
  - The Number Field Sieve
  - The Function Field Sieve

Is This Talk *EVER* Going to End?

- We can further abstract by applying a set of index calculus techniques derived from the Number Field Sieve.
- This leads to a few related approaches:
  - The Number Field Sieve
  - The Function Field Sieve
  - A few notable recent revisions to these algorithms by Joux...

Is This Talk *EVER* Going to End?

- We can further abstract by applying a set of index calculus techniques derived from the Number Field Sieve.
- This leads to a few related approaches:
  - The Number Field Sieve
  - The Function Field Sieve
  - A few notable recent revisions to these algorithms by Joux...
- As we'll see next time...

Section 3

Conclusion, Mk. I

Today's Conclusion

- The Discrete Log Problem in groups with composite order can be decomposed.
- Solving Discrete Logarithm Problems is Hard.
- There are a set of algorithms that are deterministic
  - Brute Force runs in $O(n)$ and requires little storage.
  - Baby Step, Giant Step runs in $O(\sqrt{n})$ and requires $O(\sqrt{n})$ storage.
- There are more powerful algorithms that are probabilistic
  - Pollard's ρ-method runs (heuristically, probabilistically) in $O(\sqrt{n})$ and requires little storage.
  - Index Calculus runs (probabilistically) in $L_p\left(1/2, \sqrt{2}\right)$

That's All Folks!

Thank You!

Colophon

- The principal font is Evert Bloemsma's 2004 humanist san-serif font Legato. This font is designed to be exquisitely readable, and is a significant departure from the highly geometric forms that dominate most san-serif fonts. Legato was Evert Bloemsma's final font prior to his untimely death at the age of 46.
- Math symbols are typeset using the MathTime Professional II (MTPro2) fonts, a font package released in 2006 by the great mathematical expositor Michael Spivak.
- The URLs are typeset in Luc(as) de Groot's 2005 Consolas, a monospace font with excellent readability.
- Diagrams were produced in Ti*k*Z.