The discrete logarithm problem.
2 – Subexponential algorithms

Pierrick Gaudry

CARAMEL – LORIA
CNRS, UNIVERSITÉ DE LORRAINE, INRIA

MPRI – 12.2 – 2013-2014

References

Recommended books:

- *Mathematics of Public Key Cryptography*,
  by Steven Galbraith.
  Cambridge University Press, 2012.
- *Algorithmic Cryptanalysis*,
  by Antoine Joux.
  Chapman and Hall, CRC, 2009

Plan

Refresh on smoothness

$L(1/2)$ index calculus in finite fields

An exemple of $L(1/3)$ algorithm

The L notation

Definition: subexponential L-function

Let N be the main parameter (usually the input of the algorithm). For parameters $\alpha \in [0, 1]$ and $c > 0$, we define the **subexponential** L-function by

$$L_N(\alpha, c) = \exp\left(c(\log N)^\alpha (\log\log N)^{1-\alpha}\right).$$

Rem: α is the main parameter. $\alpha = 0$ means polynomial-time; $\alpha = 1$ means purely exponential.
Rem: Sometimes, we drop the c parameter. All algorithms in this lecture will have complexity in $L_N(\frac{1}{2})$ or in $L_N(\frac{1}{3})$.
Rem: Crude approximation. The input N has $n = \log N$ bits, $L_N(\alpha) \approx 2^{n^\alpha}$.

The prime number theorem

The most "non-smooth" integers are primes. And there are tons of them!

Prime Number Theorem

Let $\pi(x)$ be the number of primes less than or equal to x. Then

$$\pi(x) \sim x/\ln(x).$$

Can be refined with the logarithmic integral:

$$\mathrm{Li}(x) = \int_2^x \frac{\mathrm{d}t}{\ln t}$$

Then we have $\pi(x) \sim \mathrm{Li}(x)$, and more precisely, under RH,

$$\pi(x) = \mathrm{Li}(x) + O(\sqrt{x}\ln x).$$

Smooth integers: examples

Pick the right answer:

- A 100-bit integer is 10-bit smooth with probability
  ☐ $1/10$ ☐ $1/5000$ ☐ $1/10^{10}$ ☐ $1/2^{60}$
- A 100-digit integer is 10-digit smooth with probability
  ☐ $1/10$ ☐ $1/5000$ ☐ $1/10^{10}$ ☐ $1/2^{60}$
- A 500-bit integer is 100-bit smooth with probability
  ☐ $1/10$ ☐ $1/5000$ ☐ $1/10^{10}$ ☐ $1/2^{60}$

Smooth integers: examples

Pick the right answer:

- A 100-bit integer is 10-bit smooth with probability
  ☐ $1/10$ ☐ $1/5000$ ☐ $1/10^{10}$ ☐ $1/2^{60}$
- A 100-digit integer is 10-digit smooth with probability
  ☐ $1/10$ ☐ $1/5000$ ☐ $1/10^{10}$ ☐ $1/2^{60}$
- A 500-bit integer is 100-bit smooth with probability
  ☐ $1/10$ ☐ $1/5000$ ☐ $1/10^{10}$ ☐ $1/2^{60}$

Hint: remember u^{-u}.

Smooth integers: theorem of CEP

Def. We let $\psi(x, y)$ be the number of y-smooth integers that are less than or equal to x.

Theorem (Canfield – Erdős – Pomerance)

For any $\varepsilon > 0$. Uniformly in $y \geq (\log x)^{1+\varepsilon}$, as $x \to \infty$,

$$\psi(x, y)/x = u^{-u(1+o(1))},$$

where $u = \log x / \log y$.

In all our algorithms, y is much larger than this bound: it is subexponential, or (in next lecture), exponential in $\log x$.

Smooth integers: theorem with L

Easy corollary of CEP:

Smoothness probabilities with L notation

Let α, β, c, d, with $0 < \beta < \alpha \leq 1$. The probability that a number less than or equal to $L_N(\alpha, c)$ is $L_N(\beta, d)$-smooth is

$$L_N\left(\alpha - \beta, (\alpha - \beta)\frac{c}{d}\right)^{-1+o(1)}.$$

Main application: $\alpha = 1$, $\beta = 1/2$.
Then an integer less than N is $L_N(1/2)$-smooth with probability in $1/L_N(1/2)$.

Smooth polynomials: theorem of PGF

Let $\mathbb{F}_q$ be a finite field. Smooth polynomials over $\mathbb{F}_q$ are exactly as frequent as smooth integers, with the **degree** taking the role of the log (both are "size" functions).

Theorem (Panario – Gourdon – Flajolet)

Let $N_q(n,m)$ be the number of monic polynomials over $\mathbb{F}_q$, of degree n that are m-smooth.
Then we have

$$N_q(n,m)/q^n = u^{-u(1+o(1))},$$

where $u = n/m$.

Rem. Degenerate case. If $m = 1$ (completely splitting polynomials), then $N_q(n,1) = q^n/n!$.

Plan

Refresh on smoothness

$L(1/2)$ index calculus in finite fields

An exemple of $L(1/3)$ algorithm

Discrete log in prime fields

Let p be a prime. Let g be a generator of $\mathbb{F}_p^*$.

Goal: compute the discrete log of h in base g.

1. Fix a **smoothness bound** B, and construct the **factor base** $\mathcal{F} = \{p_i \text{ prime}; p_i \leq B\}$.
2. Collect **relations**.
   Repeat the following until enough relations:
   2.1 Pick a and b at random and compute $z = g^a h^b$.
   2.2 Seen as an integer in $[0, p-1]$, check if z is B-smooth.
   2.3 If yes, write z as a product of elements of $\mathcal{F}$ and store the corresponding relation as a row of a matrix.
3. Find a vector v in the **left-kernel** of the matrix, modulo $p-1$.
4. Deduce the discrete log relation between h and g.

DL in prime fields: Analysis

Step 1: can be done with Erathostenes' sieve. Cost is $\widetilde{O}(B)$.
We get $\#\mathcal{F} = \pi(B) \sim B/\ln(B)$ elements.

Step 2:

The cost of computing z is $O(\log(p))$ operations in $\mathbb{F}_p$.
Testing its smoothness is more problematic:

- Computing the full factorization (with NFS) would be possible, but costly in practice.
- After Smith's lectures, you'll know about ECM, and this is the best choice asymptotically.
- Trial-division is an option if B is not too large.

We'll use NFS: the full factorization can be computed in $L_p(1/3)$.
The probability that z gives a relation is given by CEP.
We need more than $\#\mathcal{F}$ relations to ensure a non-trivial kernel.

Step 3:

The matrix is sparse; Wiedemann's algorithm gives a non-trivial kernel vector in time $\widetilde{O}((\#\mathcal{F})^2)$.

DL in prime fields: Analysis

The optimal choice for B will be of the form $L_p(\frac{1}{2}, b)$.
Then $\#\mathcal{F} = L_p(\frac{1}{2}, b + o(1))$.
The probability that z gives a relation is then

$$L_p\Big(\frac{1}{2}, -\frac{1}{2b} + o(1)\Big).$$

So the number of time we have to run the loop of Step 2 is

$$L_p\Big(\frac{1}{2}, b + o(1)\Big) \cdot L_p\Big(\frac{1}{2}, \frac{1}{2b} + o(1)\Big) = L_p\Big(\frac{1}{2}, b + \frac{1}{2b} + o(1)\Big).$$

The cost of each iteration is dominated by the factorization which, with NFS is in $L_p(\frac{1}{3})$. This is swallowed in the $o(1)$.
The value of b that minimizes Step 2 is then $b = \sqrt{2}/2$. The **total cost** of Step 2 is

$$L_p\Big(\frac{1}{2}, \sqrt{2} + o(1)\Big).$$

Finally, the cost of linear algebra is in $L_p(\frac{1}{2}, 2b + o(1))$, which is the same.

Sparse linear algebra

Theorem: Sparse kernel computation

Let M be a singular square matrix of size n, with w non-zero entries per row on average.
A uniformly distributed element of the kernel of M can be computed in time $O(wn^2)$ and memory $O(wn)$.

Rem. The corresponding algorithm works in a **black-box model**: the key operation is a matrix-vector product.
Rem. This is a probabilistic algorithm.
Rem. If the matrix is not square, then we can add empty rows/columns.

Thm. Kernel computation implies system solving.

Wiedemann's algo: min poly to kernel

Def. The **minimal polynomial** $\mu_M(T)$ of the matrix M is the non-zero polynomial of smallest degree such that $\mu_M(M) = 0$.
Rem. The characteristic polynomial is a multiple of $\mu_M(T)$. The degree of μ_M is at most n.

Hypothesis: $\mu_M(T)$ is known.
Since M is supposed to be singular, then $T \mid \mu_M(T)$.

$$\mu_M(T) = T^k \nu(T).$$

Let v be a random (non-zero) vector.
Then, $\mu_M(M)v = 0$, but with high probability $\nu(M)v \neq 0$.
Therefore, there exists $i < k$ such that $w = M^i \nu(M)v \neq 0$, and $Mw = 0$. The vector w is a non-trivial element of $\mathrm{Ker} M$.

Conclusion: given the minimal polynomial, one can get a kernel element in less than n matrix-vector products.

Wiedemann's algo: computing min poly

Let u and v be random vectors and define the **Krylov sequence**:

$$a_i = {}^t u \, M^i \, v.$$

Fact: This sequence is a linear recurrence relation sequence, whose characteristic polynomial is a factor of $\mu_M(T)$.
Hypothesis: the characteristic polynomial is equal to $\mu_N(T)$.

Thm. Given $2n$ terms of a linear recurrence relation sequence, it is possible to get its characteristic polynomial in $O(n^2)$ operations, or $\widetilde{O}(n)$ operations with fast multiplication.

Rem. Computing $2n$ terms of the sequence costs $O(n)$ matrix-vector products.
Algorithm: Extended GCD stopped in the middle.

List of **keywords** for related questions: Toeplitz linear system, rational reconstruction, Padé approximant, continued fractions.

DL in prime fields: Summary

Two key tools of the basic **index calculus**:

- Smoothness properties;
- Sparse linear algebra.

Theorem

In a prime field $\mathbb{F}_p$, the discrete logarithm problem can be solved with the basic index calculus in time $L_p(\frac{1}{2}, \sqrt{2})$.

DL in finite fields of small characteristic

Consider the DL problem in $\mathbb{F}_{2^n}$.

Dictionnary:

- Integers $\leftrightarrow$ Polynomials over $\mathbb{F}_2$.
- Primes $\leftrightarrow$ Irreducible polys.
- B-smooth $\leftrightarrow$ Degree-log B-smooth.

Then, copy-paste the same algorithm.

Discrete log in $\mathbb{F}_{2^n}$

1. Fix a smoothness bound B, and construct the factor base $\mathcal{F} = \{p_i \text{ irred}; \deg p_i \leq \log B\}$.
2. Collect relations. Repeat the following until enough relations:
   2.1 Pick a and b at random and compute $z = g^a h^b$.
   2.2 Seen as a poly of degree $< n$, check if z is smooth.
   2.3 If yes, write z as a product of elements of $\mathcal{F}$ and store the corresponding relation as a row of a matrix.
3. Find a vector v in the left-kernel of the matrix, modulo $2^n - 1$.
4. Deduce the discrete log relation between h and g.

Index calculus: general theorem

Theorem

In any finite field $\mathbb{F}_q$, the discrete logarithm problem can be solved with the index calculus algorithm in time $L_q(\frac{1}{2}, c)$.
For prime fields and fields of fixed characteristic, it is possible to choose $c = \sqrt{2}$.

Rem. This is much better than for a generic group: Pollard's Rho complexity is $\sqrt{q} \approx L_q(1, \frac{1}{2})$.
Rem. On the other hand, except for the linear algebra step, the index calculus does not take advantage of the presence of subgroups.
E.g. In the **DSA** signature scheme, we work in a Pollard-Rho-resistant subgroup of the multiplicative group of an index-calculus-resistant finite field.

Plan

Refresh on smoothness

$L(1/2)$ index calculus in finite fields

An exemple of $L(1/3)$ algorithm

Foreword about $L(1/3)$

A short history of $L(1/2)$ to $L(1/3)$ transition:

- Late 70' / early 80's, we had $L(1/2)$ algorithms for factoring and dlog in all finite fields.
- 1984: **Coppersmith**'s algorithm. $L(1/3)$ for finite fields in characteristic 2.
- End of 80's, early 90's: **Number Field Sieve** for factoring in $L(1/3)$, by Lenstra, Pollard, ...
- 1993: Gordon and Schirokauer adapted the Number Field Sieve for DL in prime fields.
- 1994: **Function Field Sieve** (Adleman). $L(1/3)$ for finite fields of small characteristic.
- 90's – mid-2000's: Improvements to NFS for DL.
- End of the 2000's: we had $L(1/3)$ algorithms for factoring and dlog in all finite fields.

Foreword about $L(1/3)$

A short history of $L(1/2)$ to $L(1/3)$ transition:

- Late 70' / early 80's, we had $L(1/2)$ algorithms for factoring and dlog in all finite fields.
- 1984: **Coppersmith**'s algorithm. $L(1/3)$ for finite fields in characteristic 2.
- End of 80's, early 90's: **Number Field Sieve** for factoring in $L(1/3)$, by Lenstra, Pollard, ...
- 1993: Gordon and Schirokauer adapted the Number Field Sieve for DL in prime fields.
- 1994: **Function Field Sieve** (Adleman). $L(1/3)$ for finite fields of small characteristic.
- 90's – mid-2000's: Improvements to NFS for DL.
- End of the 2000's: we had $L(1/3)$ algorithms for factoring and dlog in all finite fields.
- 2013: $L(1/4)$, then quasi-polynomial for small-characteristic finite fields.

Joux-Lercier's algorithm: setting

This algorithm by **Joux-Lercier** (2006) is the simplest $L(1/3)$ dlog algorithm. It works in small characteristic.

Setting:

Let $\mathbb{F}_{2^n}$ be the target finite field.

We choose an **unusual representation**: Pick γ_1 and γ_2 as follows:

- γ_1 and γ_2 are two polynomials over $\mathbb{F}_2$, of resp. degrees d_1 and d_2, with $d_1 d_2 \geq n$.
- $\gamma_1(\gamma_2(x)) - x$ has an irreducible factor φ of degree n.

Then $\mathbb{F}_{2^n} = \mathbb{F}_2[x]/\varphi(x)$.

Define $y = \gamma_2(x)$, so that in $\mathbb{F}_{2^n}$, we have

$$\begin{cases} y & = & \gamma_2(x) \\ x & = & \gamma_1(y) \end{cases}$$

From now on, x and y are **elements** of $\mathbb{F}_{2^n}$, not indeterminates.

Joux-Lercier's algorithm: factor base

The **factor base** $\mathcal{F}$ is made of two sets of elements of $\mathbb{F}_{2^n}$:

- The elements $p(x)$, where p is irreducible of degree $\leq B$;
- The elements $p(y)$, where p is irreducible of degree $\leq B$;

The **smoothness bound** B is to be set later.

The cardinality of $\mathcal{F}$ is then around $2^{B+2}/B$.

Indeed, the number of irreducible polynomials of degree n is about $2^n/n$ (very classical result, using Moebius transform).

Joux-Lercier's algorithm: relations

Consider a **bivariate polynomial** $\phi(X, Y) = A(Y) + B(Y)X$.
After evaluating ϕ at (x, y), we get two different expressions (the **norms**) for $\phi(x, y)$ in $\mathbb{F}_{2^n}$:

$$\begin{aligned}\phi(x, y) &= \phi(x, \gamma_2(x)) \\ &= \phi(\gamma_1(y), y))\end{aligned}$$

If **both** univariate expressions are B-smooth, then we get a multiplicative relation between factor base elements:

$$\prod_i p_i(x)^{e_i} = \prod_j p_j(y)^{f_j}.$$

This translates into a **linear relation** between logs of FB elements.

If one has more than $\#\mathcal{F}$ relations, then one can solve the system (assuming it has full rank).

Joux-Lercier's algorithm: analysis

Parameters: d_1, d_2, B, and $e = \deg_y \phi$.
Let's start to evaluate the degrees of the elements to test for smoothness:

$$\begin{array}{rcl} \deg \phi(x, \gamma_2(x)) & = & 1 + ed_2 \\ \deg \phi(\gamma_1(y), y) & = & d_1 + e \end{array}$$

The number of ϕ that we test is 2^{2e}. We hope to spend the **same time** testing them as doing the linear algebra whose cost is $2^{2B+o(1)}$.

So we fix $e = B$.

Let $B = e = \log_2 L_{2^n}(\frac{1}{3}, \beta)$.
We tune d_1 and d_2 to minimize the norms.

The degree of the product of the norms is then in $\log_2 L_{2^n}(\frac{2}{3}, f(\beta))$.
The probability that a ϕ gives a relation is in $1/L_{2^n}(\frac{1}{3}, g(\beta))$.

Joux-Lercier's algorithm: analysis

Expected number of relations is then $L_{2^n}(\frac{1}{3}, 2\beta g(\beta))$.
It should be larger than the factor base: $L_{2^n}(\frac{1}{3}, \beta)$.
We tune β to be as small as possible, when ensuring this inequality to hold.

Complexity of Joux-Lercier

The discrete logarithms of small elements in $\mathbb{F}_{2^n}$ can be computed in time $L_{2^n}(\frac{1}{3}, (\frac{32}{9})^{1/3})$.

Summary: what led us to $L(1/3)$ instead of $L(1/2)$?

Instead of having to smooth an element in $L(1)$, we have to simultaneously smooth two elements in $L(2/3)$.
This idea is present in **all** the $L(1/3)$ algorithms.

Joux-Lercier's algorithm: individual logs

Let h be an element for which we want the log. In general, h is not in the factor base: it has degree $\approx n$ in x.

Ignition: Let's start with the classical index-calculus.
Repeat

- Select a random integer α;
- Compute $h' = hg^{\alpha}$;
- Test whether h' is smooth.

Since we can repeat the loop only $L(1/3)$ times, we can hope for $L(2/3)$-smoothness.
We are left with the question of computing the log of elements of degree $\log_2 L_{2^n}(2/3, c)$.

Joux-Lercier's algorithm: descent

Let Q be an irreducible polynomial, s.t. we look for the log of the element $Q(x)$ (resp. $Q(y)$).

Definition: Q-lattice

The set of polynomials A_0, A_1, ..., A_k such that $\phi(X, Y) = A_0(Y) + A_1(Y)X + \cdots + A_k(Y)X^k$ verifies

$$Q(x) \mid \phi(x, \gamma_2(x)) \quad \text{or resp.} \quad Q(y) \mid \phi(\gamma_1(y), y)$$

is an $\mathbb{F}_2[X]$-lattice called the Q-lattice on the x-side (resp. on the y-side).

Lattice theory: the determinant of the Q-lattice is Q, and we can find a basis with coordinates of degree $\approx \frac{1}{k} \deg Q$.
Rem. In that case, this can be computed with an easy linear algebra, after putting indeterminates for coefficients of A_i.

Joux-Lercier's algo: analysis of descent

If $\deg Q \approx n^{\alpha}$, with $\frac{1}{3} \leq \alpha \leq \frac{2}{3}$, then:

- Take $k \approx n^{(\alpha-1/3)/2}$;
- The product of the norms has degree $\approx n^{(\alpha+1)/2}$;

In time $L_{1/3}$, we can hope to find a function ϕ such that both norms are $n^{\alpha/2+\frac{1}{6}}$-smooth.

Therefore, we can rewrite the log of $Q(x)$ (resp. $Q(y)$) in terms of the logs of smaller elements.

$$\frac{2}{3} \longrightarrow \frac{1}{2} \longrightarrow \frac{5}{12} \longrightarrow \frac{3}{8} \longrightarrow \frac{17}{48} \cdots$$

Rem. Need a careful analysis when getting close to $\frac{1}{3}$.

Joux-Lercier's algo: descent tree

A **descent tree** is constructed:

- Each node is labelled by an irreducible polynomial;
- The children of a node are such that the log of the node polynomial is a linear combination of the logs of the children polynomials.
- The degrees of the children polynomials are less than the degrees of the node polynomial.
- The arity is polynomial in n.
- The depth is polynomial in n.
- The leafs have degree less than the factor base bound.

Joux-Lercier's algo: conclusion

The complexity of the individual log step is an order of magnitude less than for the precomputation step.

Overall complexity of Joux-Lercier's algorithm

The discrete logarithm problem in $\mathbb{F}_{2^n}$ can be solved in

$$L_{2^n}\left(\frac{1}{3}, \left(\frac{32}{9}\right)^{1/3}\right).$$

This algorithm can be viewed as a special case of the Function Field Sieve (same complexity, but faster in practice).

Rem. This works not only in characteristic 2, but also in any "small" characteristic.