Analysis of discrete logarithm algorithms: arithmetic, analytic and geometric tools

Pierrick Gaudry

Caramba – LORIA
CNRS, Université de Lorraine, Inria

Aussois – March 2019

Plan

The discrete log problem in crypto

- Alice, Bob, the VPN and the blockchain...
- The discrete log problem and generic algorithms
- A few words about the quantum computer
- Pairings

Combining congruences

- Subexponential algorithms via congruences
- More about smoothness
- Overview of the current knowledge

Selected topics

- Proving the quasi-polynomial complexity?
- Proving the complexity of NFS?

Plan

Cryptographic context

Don't tell me you want yet another crypto introduction with Alice and Bob?

Cryptographic context

Don't tell me you want yet another crypto introduction with Alice and Bob?

Who has never heard about **RSA** ?

Cryptographic context

Don't tell me you want yet another crypto introduction with Alice and Bob?

Who has never heard about **RSA** ?

Who has never heard about **Diffie-Hellman** ?

Cryptographic context

Don't tell me you want yet another crypto introduction with Alice and Bob?

Who has never heard about **RSA** ?

Who has never heard about **Diffie-Hellman** ?

Who has never seen an **elliptic curve** in the wild ?

Cryptographic context

Don't tell me you want yet another crypto introduction with Alice and Bob?

Who has never heard about **RSA** ?

Who has never heard about **Diffie-Hellman** ?

Who has never seen an **elliptic curve** in the wild ?

Who has never clicked on the **small lock** in the `https://` ?

Hard problems

Public key crypto security relies on **hard algorithmic problems**.

Mainstream public key crypto

- Integer factorization (RSA);
- Discrete log problem (ElGamal enc, Schnorr sig):
  - in finite fields
  - in elliptic curves
  - in jacobians of genus 2 curves

Post-quantum crypto

- Hard problems in Euclidean lattices
- Hard problems in error correcting codes
- Paths in (supersingular) isogeny graphs
- Multivariate polynomial systems solving

Where do we find DLP over finite fields?

Examples of current usage of DLP over $\mathbb{F}_p$:

- In **VPN**s (virtual private network).
  The IKE protocol used internally relies on DLP.
- In **TLS** (used for instance in `https`).
  In order to get *forward secrecy* the session key is usually computed with Diffie-Hellman.
  DLP in prime fields is one choice among others.
  Negociation between server and client.
- In most (all ?) currently deployed **E-Voting** systems.
  ElGamal encryption is used. Most of the times with prime field. Sometimes with elliptic curves.
  Examples: Helios, Belenios, Swiss Post / Scytl.
  (note, in France, no incitation to publish the protocol)

Plan

Definition of the problem

Context: a cyclic group G of order N. Let $G = \langle g \rangle$.

Assumptions:

- there exists a fast algo for the group law in G;
- elements are represented with $\log N$ bits;
- N is known (and maybe its factorization).

Def. The **discrete logarithm problem** (DLP) in G is: given any element h, compute x such that

$$h = g^x.$$

Easy remarks

The result x makes sense only modulo N (because $g^N = 1$).

There is a group isomorphism:

$$G \cong \mathbb{Z}/N\mathbb{Z},$$

- one of the map is easy (binary exponentiation);
- the other is the DLP.

The naive algorithm can solve the DLP in less then N group operations.
$\Longrightarrow N$ must be large enough.

Pohlig-Hellman reduction

Assume the factorization $N = \prod \ell_i^{e_i}$ is known.

For any j, raise g and h to the power $N/\ell_j^{e_j}$ to obtain g' and h'. Then $x \mod \ell_j^{e_j}$ is the discrete logarithm of h' in the group of order $\ell_j^{e_j}$ generated by g'.

By **CRT**, we have therefore reduced the original DLP to smaller DLP in groups of prime powers orders.

Adding to this an **Hensel** trick, we obtained:

Theorem of Pohlig–Hellman

The DLP in G of order $N = \prod \ell_i^{e_i}$ can be reduced in polynomial time to, for each i, solving e_i DLP in subgroups of G of order ℓ_i.

Baby-step giant-step algorithm

Start again from a DLP: find x s.t. $h = g^x$.

Let us rewrite the (unknown) discrete logarithm x as

$$x = x_0 + \lceil\sqrt{N}\rceil x_1, \quad \text{where} \quad 0 \le x_0, x_1 < \lceil\sqrt{N}\rceil.$$

First phase: compute all candidate values for hg^{-x_0}; store them in an appropriate data structure.
Second phase: compute all the $g^{x_1\lceil\sqrt{N}\rceil}$ and check if there is a match.
If yes: reconstruct x from x_0 and x_1.

Complexity: $\tilde{O}(\sqrt{N})$ in time and space.

Rem. In practice, there are low-memory and parallel variants of this, (initially) due to Pollard.

Summary of generic DL algorithms

Combining Pohlig–Hellman and Baby-step giant-step, we get:

> Up to polynomial time factors, the DLP in any group can be solved in $\sqrt{\ell}$ operations, where ℓ is the largest prime factor of the group order.

The **converse is proven:**

Theorem (Shoup): Lower bound on DLP

Let A be a probabilistic generic algorithm for solving the DLP. If A succeeds with probability at least $\frac{1}{2}$ on a group G, then A must perform at least $\Omega(\sqrt{\#G})$ group operations in G.

But, of course, **no group is generic**, in the sense that the attacker is free to use a DLP algorithm specific to the family used by the designer.

Plan

Physics: a bit becomes a qubit

Def. A **qubit** is a two-state quantum-mechanical system.

Traditionally, the 2 states are denoted with the **Dirac notation**:

$$|0\rangle \text{ and } |1\rangle$$

These are the basis-elements of a 2-dimensional $\mathbb{C}$-vector space (and in fact, a Hilbert space).

A qubit is therefore a **linear combination** (called superposition)

$$|z\rangle = z_0\,|0\rangle + z_1\,|1\rangle,$$

where z_0 and z_1 are in $\mathbb{C}$ such that $|z_0|^2 + |z_1|^2 = 1$.
Observation: If one observes $|z\rangle$, 0 (resp. 1) is obtained with proba $|z_0|^2$ (resp. $|z_1|^2$).

N qubits are more than N times one qubit

Two independent qubits:

$$\begin{array}{rcl} |\varphi_a\rangle & = & a_0 |0\rangle + a_1 |1\rangle \\ |\varphi_b\rangle & = & b_0 |0\rangle + b_1 |1\rangle \end{array}$$

Two entangled qubits:

$$|\varphi_{ab}\rangle = c_{00} |00\rangle + c_{01} |01\rangle + c_{10} |10\rangle + c_{11} |11\rangle.$$

Effect of observing the first qubit:

- In the first case, does not change the probability distribution on the second qubit;
- In the second case, potentially changes the probability distribution on the second qubit.

Rem. The decoherence effect tends to transform entangled into independent: diagonalize the probabilities (this is bad).

Shor's algorithm

[There are many good descriptions available on the web]

Shor's algorithm can solve:

- Integer factorization;
- Discrete logarithm problem in any (explicit) group.

Features:

- Polynomial complexity;
- Heavily relies on the quantum Fourier transform:
  - With n qubits, perform a tranform of length 2^n.
  - Uses a quadratic number of quantum gates.
- Total number of gates is quadratic or cubic, depending how we count.

Shall we panic?

Yes, of course!

- Due to possible applications in AI, many attempts to have larger and larger prototypes;
- NIST call for post-quantum cryptography.

But:

- Shor's algorithm requires n qubits to remain entangled for a long time (n is maybe twice the bitsize of input).
  This is difficult!
- As long as there is no large quantum computer, mainstream crypto will stay.
  Changing a standard takes years or decades.
  Look at the EMV protocol.

Personal guess: RSA-1024 will be first publicly factored with a classical computer, not a quantum one.

Plan

What are pairings in crypto?

A **pairing** in crypto is a map:

$$e : G_1 \times G_2 \longrightarrow G_3,$$

where G_1, G_2, G_3 are cyclic group, and such that

- e is efficiently **computable**;
- e is **bilinear**;
- e is **non-degenerate**.

And some **problems** must be (supposedly) hard:

- Discrete logarithm problem in each of G_1, G_2, G_3;
- Inverting the pairing;
- More specific problems.

Tons of advanced crypto algorithms can be built with this tool.

Pairings in practice

Only instance: Weil pairing on **elliptic curves** (and variants).

In that case G_3 is a **finite field** of the form $\mathbb{F}_{q^k}$, where q is a prime power and k is a **small integer**.

Raises the question of the difficulty of the DLP in such extension fields.

Example of deployment: in a blockchain called ZCash, there is a "shielded" mode, to make things anonymous. Many zero-knowledge proofs have to be added. They are based on pairings (keyword here is ZK-Snarks).

Typical targets for DLP in this context: $\mathbb{F}_{p^6}$ and $\mathbb{F}_{p^{12}}$.

Summary

The finite fields **currently in use**:

- **Prime fields** $\mathbb{F}_p$, with p of 2048 bits or more;
  (but we still see way too small primes, with 768 or 1024 bits)
- **Extension fields** $\mathbb{F}_{p^6}$ or $\mathbb{F}_{p^{12}}$, with 2048 bits or more;
  Due to improvements of the last few years, need to go for larger sizes.

What about **small characteristic**? Broken!

Plan

The discrete log problem in crypto

- Alice, Bob, the VPN and the blockchain...
- The discrete log problem and generic algorithms
- A few words about the quantum computer
- Pairings

Combining congruences

- Subexponential algorithms via congruences
- More about smoothness
- Overview of the current knowledge

Selected topics

- Proving the quasi-polynomial complexity?
- Proving the complexity of NFS?

Plan

Generalities for DL in $\mathbb{F}_p$

Let G be the **multiplicative group** of $\mathbb{F}_p$, with p prime.

G is cyclic, of order $p-1$.
With Pohlig-Hellman + BSGS, we consider a subgroup of large prime order

$$\ell \mid p-1.$$

Rem. ℓ is large enough so that any event with proba $1/\ell$ is unlikely to occur.

Notation. g is a generator of the subgroup of order ℓ, and h is the target element in $\langle g \rangle$: we look for $\log_g(h)$.

A three-step strategy

Algorithm with **three phases**:

1. **Collect relations** between “small” elements;
2. With sparse **linear algebra**, deduce the logarithms of those;
3. Find a relation between the **target** h and small elements.

Rem. The first two phases depend only on $\mathbb{F}_p$. If we want the logs of many targets, these can be seen as a precomputation.

Terminology. The first phase is often called **sieve**.
Indeed, in most cases, a processus à la Erathostenes is used instead or in combination of ECM.

Smothness: definition

Smoothness

Def. An integer is B-**smooth** if all its prime factors are below B.

This is an important notion. We'll discuss it at length later.

Some French mathematician use the word "friable" instead of smooth.

Collecting relations

Fix a bound B.

Pick a **random** a, and compute g^a in $\mathbb{F}_p$.

Interpret g^a as an integer in $[1, p-1]$, and test its B-smoothness.

If yes, we obtain a **relation**; let us collect many of them:

$$g^{a_i} \equiv \prod_{q<B} q^{e_{q,i}} \mod p.$$

Taking the logarithm in base g, we get:

$$a_i \equiv \sum_{q<B} e_{q,i} \log q \mod \ell.$$

In these, the only unknown part are the $\log q$, for $q < B$: the "small" elements!

Terminology. The set of "small" elements in these algorithms is often called the **Factor base**.

Let's look at an example

Let $p = 107$, and consider DLP in the subgroup G of order $\ell = 53$. We can check that $g = 3$ is a generator.

Find a_i such that g^{a_i} is smooth:

$$\begin{aligned} g^{24} &= 5 \times 7 \\ g^{34} &= 2 \times 5 \\ g^{37} &= 2^3 \times 7 \end{aligned}$$

Taking logarithms, we get the **linear** system:

$$\begin{aligned} 24 &\equiv \quad \log(5) + \log(7) \\ 34 &\equiv \log(2) + \log(5) \\ 37 &\equiv 3\log(2) + \quad \log(7) \end{aligned}$$

Solve it mod 53 and get: $\log(2) = 25$, $\log(5) = 9$, $\log(7) = 15$.

Let's look at an example (2)

We have $p = 107$, $\ell = 53$, $g = 3$ and
$\log(2) = 25$, $\log(5) = 9$, $\log(7) = 15$.

Assume we want the discrete logarithm of $h = 19$.
We look for an exponent a such that hg^a is smooth:

$$hg^{35} \equiv 5 \times 7$$

And taking the log:

$$\log(h) \equiv \log(5) + \log(7) - 35.$$

We deduce $\log(h) = 42$.

Wait! Modulo ℓ or modulo $p-1$?

The equation for a relation:

$$a_i \equiv \sum_{q<B} e_{q,i} \log q \mod \ell.$$

is written as if elements were all in the subgroup of order ℓ.

But **they are not**! Each $q < B$ has probability $\ell/(p-1)$ to be in the subgroup $\langle g \rangle$.

Fact. The equation is **still valid**: raise the equation to $(p-1)/\ell$, take the logarithms, and divide out the result by $(p-1)/\ell$ (which is assumed to be coprime to ℓ).

Rem. Important drawback of the algorithm: even though we work in a subgroup of $\mathbb{F}_p^*$, the collection of relations can not really take advantage of that. Complexity will depend on p, not on ℓ.

Main difficulty: find smooth elements

Def. An integer is B-**smooth** if all its prime factors are below B.

Any guess of how likely it is to be smooth ?
What is the probability for a random 100-digit number to be 10-digit smooth ?

- 1% ?
- 10^{-5} ?
- 10^{-10} ?
- 10^{-50} ?

Same question with **binary** digits: probability for a random 100-bit number to be 10-bit smooth ?

Main difficulty: find smooth elements

Def. An integer is B-**smooth** if all its prime factors are below B.

Any guess of how likely it is to be smooth ?
What is the probability for a random 100-digit number to be 10-digit smooth ?

- 1% ?
- 10^{-5} ?
- 10^{-10} ?
- 10^{-50} ?

Same question with **binary** digits: probability for a random 100-bit number to be 10-bit smooth ?

Key idea

The probability of being smooth depends (almost) only on the quotient of the sizes.

Plan

Smooth numbers

Smooth numbers play a crucial role in many modern algorithms for factorization and discrete log, and more generally in algorithmic number theory.

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 usually subexponential in $\log x$.

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. Algorithms in this lecture will have complexity in $L_N(\frac{1}{2})$ or $L_N(\frac{1}{3})$.
Crude approximation. The input N has $n = \log_2 N$ bits, $L_N(\alpha) \approx 2^{n^\alpha}$.

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 $1/L_N(1/2)$.

Solving the smoothness test problem

Def. The **smoothness testing problem** is: given N and B, decide if N is B-smooth, i.e. if all its prime factor are less than B.

With trial division, can be solved in time quasi-linear in B.

The **Elliptic Curve Method** by Lenstra (1987), is better:

Complexity of ECM smoothness test (heuristic)

Given an integer N and a bound B, ECM returns either the factorization of N or fails.
If N is B-smooth, the success probability is at least $1/2$.
The running time is in $(\log N)^{O(1)} L_B(1/2, \sqrt{2} + o(1))$.

Rem. ECM as a factoring algorithm gives a worst-case complexity of $L_N(1/2, 1 + o(1))$.

Analysis of the basic DLP algorithm

Recall the **algorithm sketched on an example**.

Let p be a prime, and g be an element of order $\ell | p-1$ in $\mathbb{F}_p$.
Let $h \in \langle g \rangle$. What is $\log(h)$?
Fix a smoothness bound B.

1. **Collect relations.**
   Find many a_i's such that g^{a_i} is B-smooth.
   Write the corresponding linear equations between $\log(q)$ for primes $q < B$.
2. **Linear algebra.**
   Solve the linear system modulo ℓ and deduce all the $\log(q)$.
3. **Individual logarithm.**
   Find an element a such that hg^a is B-smooth.
   Deduce $\log(h)$.

Analysis of the basic DLP algorithm

Set $B = L_p(1/2, \sqrt{2}/2)$ for smoothness bound.

Cost of **finding a relation**: by CEP, we get $L_p(1/2, \sqrt{2}/2 + o(1))$.

Cost of building the **whole matrix**: $L_p(1/2, \sqrt{2} + o(1))$.

Cost of **linear algebra**: this is sparse, over $\mathbb{F}_\ell$, so again $L_p(1/2, \sqrt{2} + o(1))$.

Once we know the values of the $\log q$'s, we can find a **single relation involving the target**: $hg^a \equiv \prod_{q<B}(\log q)^{e_q}$, in time $L_p(1/2, \sqrt{2}/2 + o(1))$.

Hence, the **total time** is

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

Combining congruences for DL in $\mathbb{F}_{2^n}$.

Representation of the finite field:

$$\mathbb{F}_{2^n} \cong \mathbb{F}_2[t]\varphi(t),$$

where $\varphi(t)$ is irreducible of degree n.

Exactly the **same algorithm**, based on the **smoothness of polynomials**:

Def. A polynomial in $\mathbb{F}_2[t]$ is b-smooth if all its irreducible factors have degree at most b.

Analogies with integers:

- Size: logarithm $\leftrightarrow$ degree;
- Number of irreducible polynomials $\approx$ number of prime numbers;
- Test of smoothness can be done in polynomial-time (don't need complicated algorithms like ECM).

Analysis of the algorithm

The probability of smoothness is very similar to the integer case:

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 $u = n/m$.

Setting a smoothness bound of $b = \log_2 L_{2^n}(1/2, \sqrt{2}/2)$, we get a total complexity of

$$L_{2^n}(1/2, \sqrt{2} + o(1)).$$

Plan

The discrete log problem in crypto

Combining congruences

- Subexponential algorithms via congruences
- More about smoothness
- Overview of the current knowledge

Selected topics

Which algorithm?

Fact: The basic combining of congruences in $L(1/2)$ works for **any finite field**.

- Small characteristic: smoothness of polynomials.
- Large characteristic: smoothness of integers.

$L(1/2)$ complexity can be **proven**.

With the **NFS/FFS algorithms**, we can get an (heuristic) $L(1/3)$ algorithm for any finite field.
(Latest hard case, in $\mathbb{F}_{p^n}$ when $n \approx \log p$, was solved in 2007).

With the **quasi-polynomial techniques** (2013-), we can go much faster in small characteristic.

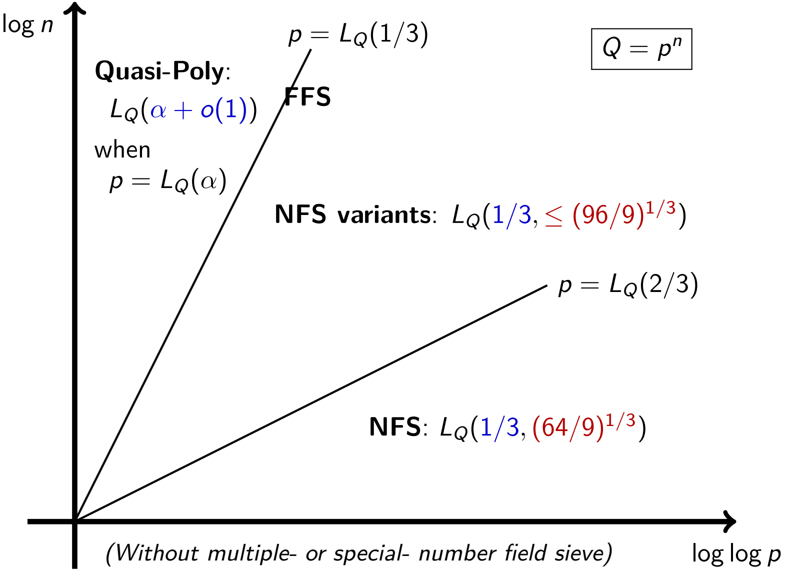
Which algorithm?

DLP in $\mathbb{F}_q$, where $q = p^n$.

- **Quasi-polynomial algorithms.** ($\approx$ 2013)
  Well suited for small characteristic (including 2).
  Complexity can be as low as $\log(q)^{O(\log\log(q))}$.
- **Number Field Sieve (NFS).** (early 90's)
  Well suited for large characteristic (including prime fields).
  Can be adapted for medium characteristic.
  Complexity in $L_q(1/3)$.
- **Function Field Sieve (FFS).** (90's)
  Still the best for a corner case of medium characteristic.
  Complexity in $L_q(1/3)$.

Sad truth: None of these complexities are fully proven.

Best current known complexities (heuristic)



Plan

The discrete log problem in crypto

- Alice, Bob, the VPN and the blockchain...
- The discrete log problem and generic algorithms
- A few words about the quantum computer
- Pairings

Combining congruences

- Subexponential algorithms via congruences
- More about smoothness
- Overview of the current knowledge

Selected topics

- Proving the quasi-polynomial complexity?
- Proving the complexity of NFS?

Plan

The discrete log problem in crypto

Combining congruences

Selected topics

- Proving the quasi-polynomial complexity?
- Proving the complexity of NFS?

How to get a quasi-polynomial complexity?

Note: Version presented is by Granger–Kleinjung–Zumbrägel (2018).

Key point: assume the field has a nice subfield representation $\mathbb{F}_{q^{4k}}$ given as

$$\mathbb{F}_{q^{4k}} \subset \mathbb{F}_{q^4}[X]/(h_1(X)X^q - h_0(X)),$$

where h_0 and h_1 have degree ≤ 2 and there exists an irreducible factor $I(X)$ of degree k in $h_1(X)X^q - h_0(X)$.

Goal: Rewrite all elements in terms of linear polynomials over $\mathbb{F}_{q^4}$.

Important remark. If $k \approx q$, then q is *polynomial* because the input size $\approx q \log q$.

Quasi-polynomial complexity is $q^{O(\log q)}$.

Elements of the finite field

$$\mathbb{F}_{q^{4k}} \subset \mathbb{F}_{q^4}[X]/(h_1(X)X^q - h_0(X))$$

Elements are represented as **polynomials over** $\mathbb{F}_{q^4}$.

Building block 1: deg 2 to linear

Let $Q \in \mathbb{F}_{q^4}[X]$ be an **irreducible polynomial of degree 2**.
Consider the set of polynomials, for a, b, c in $\mathbb{F}_{q^4}$:

$$P_{a,b,c} = X^{q+1} + aX^q + bX + c = X^q(X + a) + bX + c$$

that, after mapping X^q to h_0/h_1, become divisible by Q.

Then:

- The probability that $P(X)$ splits in linear factors is in $1/q^3$;
- The probability that its becomes divisible by Q after the transformation is in $1/q^8$.

There are q^{12} choices: we should find one in time $\approx q$.

Rem. If we start with $\mathbb{F}_{q^r}$ instead of $\mathbb{F}_{q^4}$, we expect $\approx q^{r-3}$ winners among q^{3r} choices.

Building block 1: deg 2 to linear (cont'd)

Find a, b, c in $\mathbb{F}_{q^4}$ such that

$$X^{q+1} + aX^q + bX + c = X^q(X + a) + bX + c \equiv \frac{h_0}{h_1}(X + a) + bX + c$$

splits completely on the LHS and is divisible by Q on the RHS.

We get a linear relation between logs of Q and linear elements.

Proving this can be done by studying the number of points on a (singular) plane curve. Original proof by GKZ. Simpler proofs by Göloğlu-Joux, and by Kleinjung-Wesolowski **[Talk of Thursday evening]**

Rem. Need to replace $\mathbb{F}_{q^4}$ by $\mathbb{F}_{q^{18}}$.

Building block 2: view deg $2d$ as deg 2

Let $Q \in \mathbb{F}_{q^4}[X]$ be an **irreducible polynomial of degree** $2d$.

Over $\mathbb{F}_{q^{4d}}$, Q is a product of d polynomials of degree 2.

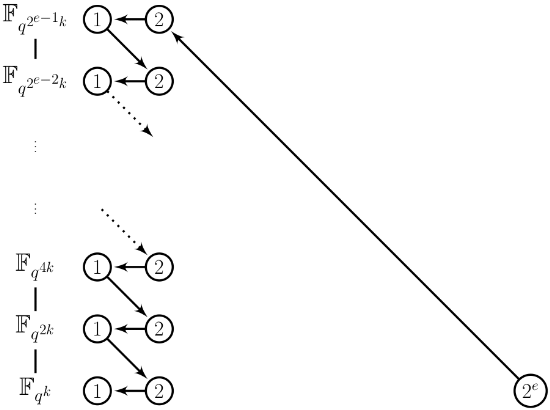
For each factor Q' of degree 2, apply building block 1 to Q': rewrite it with **linear polynomials** over $\mathbb{F}_{q^{4d}}$.

Then go down with the **norm map**: linear over $\mathbb{F}_{q^{4d}}$ becomes degree d over $\mathbb{F}_{q^4}$.
(and irreducible factors divide d.)

Putting things together

- First, **randomize** the target element to see it as an irreducible polynomial of degree a power of 2 (anti-smoothing!).
- Then, **apply** building block 2 **recursively**, since it produces only polynomials of degree a power of 2.
- In the end, **get a linear relation** between the logs of the target and the linear polynomials over $\mathbb{F}_{q^4}$.
- **Repeat** q^4 times to be able to eliminate the logs of the linear polynomials and conclude!

With a picture



[*picture from* On the powers of 2, *by Granger, Kleinjung, Zumbrägel*]

Last piece of non-proven step

Everything can be made **rigorously proven** except for the existence of the **nice field representation**.

Furthermore, this works incredibly well **in practice**!

Still, we already have:

> **Thm.** *(Granger, Kleinjung, Zumbrägel) For every fixed p, there exist infinitely many extension fields $\mathbb{F}_{p^n}$ for which the DLP in $\mathbb{F}_{p^n}$ can be solved in expected quasi-polynomial time.*

Rem. Even when the extension degree n is prime, no practical problem to find an appropriate extension with the nice representation.

Proving the descent phase

This step was not proven in the original proposals of quasi-polynomial algorithms.

- First proof by Granger Kleinjung Zumbrägel (2014).
  Complicated plane curve with a strong role of $\mathrm{PGL}_2(\mathbb{F}_q)$.
  Proof is a bit tedious, with several sub-cases to study.
- Recent preprint by Göloğlu and Joux
  After various algebraic manipulations, obtain a much simpler curve, easier to analyse.
- Kleinjung Wesolowski (2018)
  Curve constructed in a much more elegant way. But require more algebraic gemoetry background to understand the proof.
  [Talk of Thursday evening]

In all these proofs: show that the curve is irreducible, apply Weil's bound, deduce there are enough points, i.e. solutions to the initial problem.

(not) Proving the field representation

Recall the missing part to get a fully proven algorithm:

Missing result (unproven)

For any finite field $\mathbb{F}_q$, for any integer $k \leq q + 2$, there exists an integer $d \in O(\log q))$ and h_0,h_1, two polynomials in $\mathbb{F}_{q^d}[X]$ of degree at most 2 such that

$$h_1(X)X^q - h_0(X)$$

has an irreducible factor of degree k.

Unclear how hard this problem is.

(not) Proving the field representation

Recent paper by Giacomo Micheli.
On the selection of polynomials for the DLP quasi-polynomial time algorithm in small characteristic

The idea is to use for h_0 and h_1 some **specific polynomials** with just **one** free parameter t.

Then $F(t, X) = h_1(t, X)X^q - h_0(t, X)$ defines a field extension of $\mathbb{F}_q(t)$, and **Chebotarev Density Theorem** tells the probability to obtain a given factoring pattern for $F(t_0, X)$ for a random value of t_0.

The answer depends a lot on the **Galois group** of $F(t, X)$.

Chebotarev density theorem for number fields

Chebotarev density theorem

Let K be a number field that is Galois over $\mathbb{Q}$. Let $H \subset \mathrm{Gal}(K/\mathbb{Q})$ be a conjugacy class. Then

$$\mathrm{Prob}(\mathrm{Frob}(p) = H) = \frac{\#H}{\#\mathrm{Gal}(K/\mathbb{Q})}.$$

Here, $\mathrm{Frob}(p)$ is defined as follows:

- Consider all prime ideal $\mathfrak{p}$ above p;
- Let $\mathrm{Dec}(\mathfrak{p})$ be the subgroup of $\mathrm{Gal}(K/\mathbb{Q})$ that stabilizes $\mathfrak{p}$.
- There is a morphism to the Galois group of the residue field:

$$\alpha_{\mathfrak{p}} : \mathrm{Dec}(\mathfrak{p}) \to \mathrm{Gal}(K_{\mathfrak{p}}/\mathbb{F}_p).$$

- Consider the preimages of the Frobenius automorphism of $K_{\mathfrak{p}}$.
- The union of those is a conjugacy class called $\mathrm{Frob}(p)$.

Simple application

We assume that we are in the generic case:

Let $f(x) \in \mathbb{Z}[x]$ be an irreducible polynomial of degree d, such that its Galois group is the **full symmetric group**.

Then, applying the theorem to the Galois closure of the extension generated by $f(x)$, we get

- The probability that $f(x)$ splits completely modulo a prime p is $1/d!$;
- The probability that $f(x)$ stays irreducible modulo a prime p is $1/d$.

An exercise

Galois $\mathbb{Z}/2 \times \mathbb{Z}/2$:

Let $f(x) = x^4 + 1$. Its Galois group is $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z}$.
Applying the theorem to $K = \mathbb{Q}[x]/f(x)$, reducing modulo a prime p, we get:

- The probability that $f(x)$ splits completely is $1/4$;
- The probability that $f(x)$ has 2 irreducible factors of degree 2 is $3/4$;
- The other cases can not occur.

Back to Micheli's (not-)proof

Chebotarev density theorem for **function fields** is slightly more involved due to fields of constants.

But Micheli proves that for his choice of $h_0(t, X)$ and $h_1(t, X)$, the Galois group of $h_1(t, X)X^q - h_0(t, X)$ is the **full symmetric** group.

He deduces that all the degrees can occur after randomizing t.

Unfortunately, there is **no control** on the degree d of the field extension $\mathbb{F}_{q^d}$ where t is going to lie.

Remember, we would need $d \in O(\log q)$.

Plan

The discrete log problem in crypto

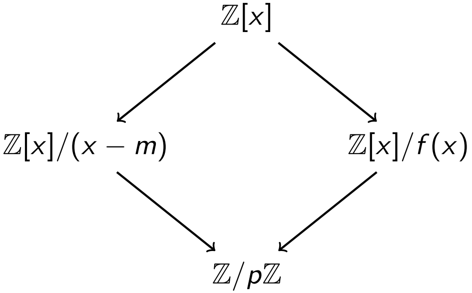
Combining congruences

Selected topics

- Proving the quasi-polynomial complexity?
- Proving the complexity of NFS?

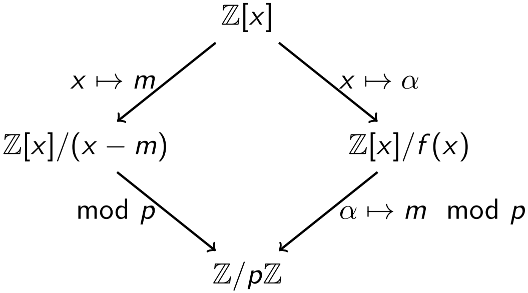
The NFS diagram for DLP in $\mathbb{F}_p^*$

Let $f(x)$ a polynomial and m integer, such that $f(m) \equiv 0 \mod p$.



The NFS diagram for DLP in $\mathbb{F}_p^*$

Let $f(x)$ a polynomial and m integer, such that $f(m) \equiv 0 \mod p$.



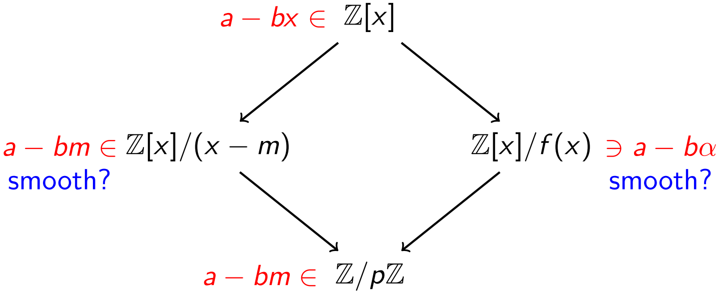
The NFS diagram for DLP in $\mathbb{F}_p^*$

Let $f(x)$ a polynomial and m integer, such that $f(m) \equiv 0 \mod p$.

$$
\begin{array}{ccccc}
 & & a - bx \in \mathbb{Z}[x] & & \\
 & \swarrow & & \searrow & \\
a - bm \in \mathbb{Z}[x]/(x - m) & & & & \mathbb{Z}[x]/f(x) \ni a - b\alpha \\
 & \searrow & & \swarrow & \\
 & & a - bm \in \mathbb{Z}/p\mathbb{Z} & &
\end{array}
$$

The NFS diagram for DLP in $\mathbb{F}_p^*$

Let $f(x)$ a polynomial and m integer, such that $f(m) \equiv 0 \mod p$.



If both sides are smooth, linear **relation** between logs in $\mathbb{Z}/p\mathbb{Z}^*$.

Rem. Enough to have smooth "norms": $f(a/b)b^{\deg f}$ and $a - bm$.

NFS: main steps

1. **Polynomial selection**: choice of f and m.
2. **Collecting relations**: find (a, b)-pairs such that both sides are smooth.
3. Prepare the matrix (ugly details hidden).
4. **Linear algebra**: get a kernel vector modulo $\ell | p - 1$.
5. **Individual log**: rewrite the log of the target in terms of logs of factor base elements.

In practice: Steps 2. and 4. are the most time-consuming.

Polynomial selection

Goal: Find f, g s.t. $p | \mathrm{Res}(f, g)$ and resulting norms $f(a/b)b^{\deg f}$ and $g(a/b)b^{\deg g}$ are as small as possible.

Base-m construction: take $m \approx p^{1/(d+1)}$, where $d \approx (\frac{\log p}{\log\log p})^{1/3}$.
Write $p = f_0 + f_1 m + f_2 m^2 + \cdots + f_d m^d$, with $0 \leq f_i < m$.
Take $g = x - m$ and $f = f_0 + f_1 x + f_2 x^2 + \cdots + f_d x^d$.
Note: many practical improvements. See Kleinjung (2006), Bai, Bouvier, Kruppa, Zimmermann (2016). Usually in the context of factorization.

Joux-Lercier construction: Use the fact that p is prime.
Consider the lattice of polynomials with a given root modulo p and use lattice reduction.
Same complexity in the end, but better in practice.

Both norms are $\approx L_p(2/3, \ldots)$.

Collecting relations

Pick (a, b) and check if both norms are **simultaneous smooth**.
If yes, this gives a **linear relation** between logarithms of small elements.

Complexities:

- a and b around $L_p(1/3)$;
- Norms are about $L_p(2/3)$;
- Smoothness bound set to $L_p(1/3)$;
- CEP theorem: probability of smoothness is $L_p(1/3)^{-1}$.
- From this, deduce the overall $L_p(1/3)$ complexity.

Key of NFS speed: instead of waiting for the smoothness of one element of size $L_p(1)$, we hope for two elements of size $L_p(2/3)$ to be smooth.

NFS and the smoothness question

Big caveat in the analysis:

> We assume that the probability for a norm to be smooth is the same as for a **random integer** of the same size.

Proving this is hard !

But there have been **interesting progress** in recent years.

Some results of Lachand

Armand Lachand (2015) started to prove smoothness results in the direction we want.

Thm. (approximate statement). For f, a polynomial of degree 2, the probability that $b^2 f(a/b)$ is smooth is about the same as expected, when a and b are chosen in a (growing) rectangle.

But in NFS:

- There are 2 sides that must be simultaneously smooth;
- The polynomial f can have arbitrary large degree.

Some results of Lachand

Lachand also proved:

Thm. (approx. statement). Let $f(x) = x^3 + 2$. The probability that $b^3 f(a/b)$ is smooth is about the same as expected, when a and b are chosen in a (growing) rectangle.

The proof is a dense, 50-page long article that mixes high-tech tools from analytic number theory.

The **conclusion** of Lachand's work is that current knowledge is probably **not yet ready** for proving NFS as we use it.

Analysis of a randomized NFS

As usual: if you can't prove an algorithm, **change it** to a variant that is easier to prove !
Impressive work in this direction:

> Jonathan D. Lee and Ramarathnam Venkatesan. *Rigorous analysis of a randomised number field sieve.* Journal of Number Theory 2018)

Theorem

There exists a variant of NFS that, given an integer N to factor, produces two integers x and y such that

$$x^2 \equiv y^2 \mod N$$

in expected time $L_N(1/3, ((64/9)^{1/3} + o(1)))$.
Heuristically, there is a good chance that $x \not\equiv \pm y \mod N$.

Analysis of a randomized NFS

$$x^2 \equiv y^2 \mod N$$

Heuristically, the two integers x and y are independent, so that this produces a non-trivial factorisation of N with probability at least $1/2$.

Indeed, if $x \not\equiv \pm y \mod N$, then compute

$$\mathrm{GCD}(x - y, N).$$

The **proof** of the previous theorem contains several parts also apply to NFS for **discrete logarithm** in $\mathbb{F}_p$.

The modified algorithm

1. **Generate** $L_N(1/3)$ polynomials $f_i(x)$, sharing the same m as a root modulo N.
2. **Collect** (a, b) pairs for all of them, in a parallel way.
3. In none of the polynomials gets enough relations, **start again**.
4. Pick one f_i for which we have enough relations, and **finish with the classical NFS**.

Main result. With good probabilities, **at least one** of the $f(x)$ has good smoothness properties, so that the failure in step 3. will rarely occur.

Note: We skip details about how to deal rigorously with algebraic obstructions (20 pages in the paper), because it does not translate to DLP.

Key for proving the smoothness

Idea. The family of f_i's is chosen so that, for a given (a, b), the probabilities of the smoothness of $b^d f_i(a/b)$ can be **analyzed simultaneously**.

Many difficult details to solve, in order to keep the same complexity as the usual NFS.

In particular, the same approach allows to study the simultaneous smoothness of both sides.

Additional algorithmic trick in order to avoid having to compute the second-moment of the probabilities.

Rigorously testing the smoothness

Now that we know that they **exist**, how do we **detect efficiently** the smooth pairs?
Usual answer: ECM. But this is not rigorously proven.

Possible solutions:

- Follow Pomerance and use an **average analysis of ECM**(average on many numbers to test);
- Use **HECM**: a variant with genus-2 curves that can be proven to detect smooth numbers efficiently.

Ref: Lenstra, Pila, Pomerance. *A hyperelliptic smoothness test, I*. 1993.

Lee-Venkatesan result: summary

Let's recap:

They **prove** that for the randomized NFS:

- The smoothness can be analyzed, so that the relation collection works as expected;
- The algebraic obstructions due to units and class groups can be controlled rigorously.

The first part **applies directly** to NFS for DLP.

The second part would have **to be adapted** for DLP (no idea how hard it would be).

The final step (non-trivial congruence vs individual logarithm) is **missing** in both facto and DLP.

Conclusion

Proving discrete log algorithms requires:

- arithmetic, discrete maths;
- algebraic number theory;
- analytic number theory;
- algebraic geometry;
- ...

And more often than not, the **algorithm must be changed** to become easier to analyze.

Will we have soon a better than $L(1/2)$ proven complexity for DLP?