Integer factorization and discrete logarithm problems

Pierrick Gaudry

Caramel – LORIA
CNRS, Université de Lorraine, Inria

JNCF – CIRM – November 2014

Plan

Presentation of the problems
- Cryptographic background
- Primality, smoothness, factorization
- The discrete logarithm problem

Combining congruences
- Basic subexponential factoring algorithm
- Combining congruences for DLP

DLP in finite fields of small characteristic
- The BaGaJoTh quasi-polynomial DLP algo
- Practice: $L(1/4)$ algo by Joux

The linear algebra step of the number field sieve
- Overview of the NFS algorithm
- Linear algebra: the filtering step

Plan

Presentation of the problems

- Cryptographic background
- Primality, smoothness, factorization
- The discrete logarithm problem

Combining congruences

DLP in finite fields of small characteristic

The linear algebra step of the number field sieve

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://` ?

What's in the lock?

When clicking on the lock in Firefox or Chromium:

- Information about **certificates**: almost all are RSA-based;
- Information about the **connection**: wide choice of algorithms.

Thanks to `Heartbleed`, many web-servers have been upgraded recently; and now they support elliptic curves!

Problem: Certificates must be understand by **all** the clients, so

RSA is here to stay.

Plan

Presentation of the problems

- Cryptographic background
- Primality, smoothness, factorization
- The discrete logarithm problem

Combining congruences

DLP in finite fields of small characteristic

The linear algebra step of the number field sieve

Basic definitions

Def. The **integer factorization problem** is: given N, compute its decomposition in prime factors $N = \prod p_i^{e_i}$.

Def. The **primality testing problem** is: given N, decide if N is a prime or a composite number.

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.

Primality is easy!

Fact: Proving that a number is composite is **very easy.**

Use **Miller-Rabin** (some kind of Fermat's little theorem: if there exists a such that $a^{N-1} \not\equiv 1 \mod N$, the N is not prime).

This is difficult to turn that into an algorithm **proving primality**.

Two approaches:

- With **elliptic curves**: Las Vegas algorithm that works in polynomial time (needs genus 2 curves as well to get a rigorous proof of the expected runtime);
- With **AKS**: polynomial deterministic.

In **practice**: be happy with a Monte Carlo algorithm, or use elliptic curves.

Listing primes is also easy

Prime Number Theorem

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

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

Fact: Listing all primes up to B can be done in **quasi-linear time** in B.
Rem. The size of the ouput has $\approx B$ bits.

Algorithm: sieve of Erathostenes.

There has been advances in the past decades: save $\log\log$ factors, save memory, improve practicality (on-line algorithm).

Exercise: implement a quasi-linear Erathostenes on a Turing machine.

Factorization by trial division

Trial division algorithm:
Try to divide N by all the primes in increasing order, until the quotient is itself a prime.
Complexity: quasi-linear in the **second largest prime factor** of N.
Worst case: $\tilde{O}(\sqrt{N})$.

Numbers easy to factor

Fact. Integers for which the second largest prime divisor is polynomial can be factored in polynomial-time.

Counting them, we realize that they are plenty of those (this includes all the primes!)

Smooth numbers

Smooth numbers play a crucial role in many modern algorithms for factorization and discrete log.

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 $L_N(\frac{1}{2})$, $L_N(\frac{1}{3})$ or $L_N(\frac{1}{4})$.

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 $L_N(1/2, 1 + o(1))$.

Summary

- Primality: easy.
- Factorization: hard.
- Smoothness test: in between.

Plan

Presentation of the problems

- Cryptographic background
- Primality, smoothness, factorization
- The discrete logarithm problem

Combining congruences

DLP in finite fields of small characteristic

The linear algebra step of the number field sieve

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.

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

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

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

Presentation of the problems
- Cryptographic background
- Primality, smoothness, factorization
- The discrete logarithm problem

Combining congruences
- Basic subexponential factoring algorithm
- Combining congruences for DLP

DLP in finite fields of small characteristic
- The BaGaJoTh quasi-polynomial DLP algo
- Practice: $L(1/4)$ algo by Joux

The linear algebra step of the number field sieve
- Overview of the NFS algorithm
- Linear algebra: the filtering step

Plan

Two congruent squares

Let N an integer to be factored.
Assume that we have found X and Y such that

$$X^2 \equiv Y^2 \mod N,$$

in a non-trivial manner: $X \not\equiv \pm Y \mod N$. Then

$$\boxed{\mathrm{GCD}(X - Y, N) \text{ gives a proper factor of } N.}$$

For a fixed value of Y, how many X's such that $X^2 - Y^2 \equiv 0$?

- If N is a prime, exactly 2 values: $\pm Y$;
- If $N = pq$ is an RSA key, 2 values mod p and 2 values mod q. By CRT, any 2 can be combined: 4 choices;
- In general, even more choices.

Fact. Computing a modular square root is as hard as factoring.

Mixing squares and smoothness

Pick x a random element modulo N.

Compute $x^2 \mod N$ as an integer in $[0, N-1]$, if we recognize a square integer, then we win.

But this is **highly unlikey**.

Therefore, we test it for B-**smoothness**, and if it is smooth, keep it for later use.

Let us do this many times, and collect many **relations**:

$$x_i^2 \equiv \prod_{p<B} p^{e_{p,i}}.$$

Combining relations to make a square

Example: $B = 11$.
If we have

$$\begin{aligned} x_1^2 &\equiv 2^2 \times 3^0 \times 5^2 \times 7^1 \mod N \\ x_2^2 &\equiv 2^0 \times 3^2 \times 5^1 \times 7^0 \mod N \\ x_3^2 &\equiv 2^1 \times 3^1 \times 5^1 \times 7^0 \mod N \\ x_4^2 &\equiv 2^2 \times 3^3 \times 5^1 \times 7^1 \mod N \\ x_5^2 &\equiv 2^3 \times 3^1 \times 5^0 \times 7^1 \mod N \end{aligned}$$

Then, we can try to select a **subset of relations** whose product is a square on the RHS.

Rem. Only the **parity** of the exponent is important.
On this example:

$$(x_1 x_2 x_3 x_5)^2 \equiv 2^6 \times 3^4 \times 5^4 \times 7^2 \mod N.$$

And we can try to factor N with:

$$\mathrm{GCD}(x_1 x_2 x_3 x_5 - 2^3 \times 3^2 \times 5^2 \times 7^1, N)$$

Combining relations with linear algebra

Let us form the **relation matrix** M:

- each row encodes a relation;
- each column is labelled by a prime $< B$;

Finding the appropriate combination is a left-kernel computation of M, seen over $\mathbb{F}_2$ (only parity is important).

Rem. The matrix is sparse, even if B is large: at most $\log N$ non-zero entries.

Example: The matrix for the RSA-768 computation was with 64G rows and 40G cols.

An $L_N(1/2)$ factoring algorithm

Tuning smoothness bound B:

- Too large: will need many relations to get a matrix with a non-zero kernel vector;
- Too small: very unlikely to get a B-smooth element.

$$\boxed{\text{The optimal value is } B = L_N(1/2, \sqrt{2}/2).}$$

Probability of being smooth:
According to CEP, it is in $L_N(1/2, \sqrt{2}/2 + o(1))^{-1}$.

Total cost of getting enough relations:
Need more relations than unknowns, say B. Testing smoothness is done in time $B^{o(1)}$ with ECM.
Therefore, total cost of constructing M is $L_N(1/2, \sqrt{2} + o(1))$.

Cost of linear algebra:
With Wiedemann or Lanczos, quasi-quadratic time:
$L_N(1/2, \sqrt{2} + o(1))$ as well.

An $L_N(1/2)$ factoring algorithm

Global complexity:

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

Trick to reduce the complexity: don't take random x's when looking for relations, but take $x = \lceil \sqrt{N} \rceil + \varepsilon$.

Therefore, $x^2 - N \approx \sqrt{N}$ instead of N before, and the probability of being smooth is higher.

Exercise: re-tune N and show that this gives a heuristic complexity of $L_N(1/2, 1 + o(1))$.

Rem. Proving those complexity is **hard** (don't try this at home...), but feasible (Lenstra, Pomerance, ...)

Factoring: going further

The **Number Field Sieve** algorithm (NFS) is another way to create relations.

- Invented in the 90's by Pollard, Lenstra, and others...
- Uses number fields (!)
- Complexity is (heursitic):

$$L_N(1/3, \sqrt[3]{64/9} + o(1)).$$

- Faster than other algorithms for integers with more than ≈ 100 decimal digits.

Rem. Will discuss this algorithm in the context of DLP in prime fields at the end of this lecture.

Plan

Presentation of the problems

Combining congruences

- Basic subexponential factoring algorithm
- Combining congruences for DLP

DLP in finite fields of small characteristic

The linear algebra step of the number field sieve

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.

Collecting relations

[Very similar to integer factorization.]

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**.

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.

Final analysis

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

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

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

DLP: going further

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

It 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).

Plan

Presentation of the problems
- Cryptographic background
- Primality, smoothness, factorization
- The discrete logarithm problem

Combining congruences
- Basic subexponential factoring algorithm
- Combining congruences for DLP

DLP in finite fields of small characteristic
- The BaGaJoTh quasi-polynomial DLP algo
- Practice: $L(1/4)$ algo by Joux

The linear algebra step of the number field sieve
- Overview of the NFS algorithm
- Linear algebra: the filtering step

Plan

Presentation of the problems

Combining congruences

DLP in finite fields of small characteristic

- The BaGaJoTh quasi-polynomial DLP algo
- Practice: $L(1/4)$ algo by Joux

The linear algebra step of the number field sieve

Setting

The BaGaJoTh algorithm applies to finite field of a **specific form:**

Definition

A finite field admits a **sparse medium subfield representation** if it can be written $\mathbb{F}_{q^{2k}}$, with

- $k \leq q + 2$;
- There exist h_0 and h_1 of degree 2 in $\mathbb{F}_{q^2}[X]$ such that $h_1(X)X^q - h_0(X)$ has an irreducible factor $\varphi(X)$ of degree k.

All finite fields of small characteristic have "more or less" a sparse medium subfield representation:

- Might need to **embed** into a larger field (more on that later);
- Might need to use polynomials h_0 and h_1 of (constant?) **degree larger** than 2.

Sizes – complexities

The algorithm behaves well if $q \approx k$.

In that case:

- The field has cardinality $q^{2k} \approx q^q$;
- The input bit-size is $\approx q \log q$;
- Any step with a time-complexity in $q^{O(1)}$ is polynomial;
- In the end, we won't reach a polynomial-time complexity, so no need to keep track of the constants in the exponent.

Rem. Quasi-polynomial means here $q^{O(\log q)}$.

Rem. Heuristic means we do not know how to prove the complexity (see work of Granger, Kleinjung, Zumbrägel, though!)

One descent step

Elements are represented by **polynomials** in $\mathbb{F}_{q^2}[X]$ of degree $< k$.
Let $P(X)$ of degree $D < k$ be an element whose log is wanted.
We are going to rewrite $\log P$ in terms of logarithms of elements of degree at most $D/2$.

Key equation:

$$X^q - X = \prod_{\alpha \in \mathbb{F}_q} X - \alpha,$$

seen as a polynomial equation in $\mathbb{F}_{q^2}[X]$.

Main idea: Replace X by $(aP + b)/cP + d)$ in the key equation, for a, b, c, d in $\mathbb{F}_{q^2}$, and hope to get a relation.

$$(aP+b)^q(cP+d) - (aP+b)(cP+d)^q = \prod_{(\alpha:\beta) \in \mathbb{P}^1(\mathbb{F}_q)} (-c\alpha + a\beta)P - (d\alpha - b\beta),$$

Let us **study both sides**.

One descent step: LHS

$$\mathrm{LHS} = (aP+b)^q(cP+d) - (aP+b)(cP+d)^q = \cdots$$

This has **high degree**, but can be rewritten modulo $\varphi(X)$:
For instance,

$$(aP+b)^q \equiv a^q \tilde{P}(h_0/h_1) + b^q \mod \varphi,$$

where $\tilde{P}$ is P with coefficients raised to the power q.

Need to clear denominators: multiply by h_1^D.
Then the degree is $2D$ for this block, and finally:

$$\boxed{\deg \mathrm{LHS} = 3D.}$$

Hope: This splits into factors of degree $D/2$.
This occurs with a **constant** probability.

One descent step: RHS

$$\mathrm{RHS} = \prod_{(\alpha:\beta)\in\mathbb{P}^1(\mathbb{F}_q)} (-c\alpha + a\beta)P - (d\alpha - b\beta)$$

Without transformation: this is **already factored**!

All factors (up to constant) belong to **translates** of P:

$$\left\{ P + \gamma \ : \gamma \in \mathbb{F}_{q^2} \right\}.$$

Rem. Not small degree, but small number of elements.

Let a, b, c, d vary...

Problem. Many quadruples give the same relation.
The appropriate structure is

$$\mathcal{P} = \mathrm{PGL}_2(\mathbb{F}_{q^2})/\mathrm{PGL}_2(\mathbb{F}_q).$$

Fact. Taking for a, b, c, d, one rep. in each coset of $\mathcal{P}$, we avoid obvious duplicates.

Number of choices:

$$\#\mathcal{P} = q^3 + q.$$

Probability that LHS is $\frac{D}{2}$-smooth: heuristically, constant.

Number of survivors: $\Theta(q^3)$ relations that involve

- polynomials of degree at most $D/2$ on the LHS;
- translates of P on the RHS.

One descent step: conclusion

Assumption: the system of linear equations formed by the surviving RHS's has **full rank**.
With a linear algebra step, we can eliminate all the unknowns on the RHS but P.

Result:

Proposition (heuristic)

Let $P(X) \in \mathbb{F}_{q^2}$ of degree $D < k$. In time polynomial in D and q, we can express $\log P$ as a linear combination $\sum e_i \log P_i$, where $\deg P_i \leq D/2$, and the number of P_i is in $O(q^2 D)$.

The descent tree

Each node of the descent tree corresponds to one application of the Proposition, hence its arity is in $q^2 D$.

level	$\deg P_i$	width of tree
0	k	1
1	$k/2$	$q^2 k$
2	$k/4$	$q^2 k \cdot q^2 \frac{k}{2}$
3	$k/8$	$q^2 k \cdot q^2 \frac{k}{2} \cdot q^2 \frac{k}{4}$
$\vdots$	$\vdots$	$\vdots$
$\log k$	1	$\leq q^{2 \log k} k^{\log k}$

Total number of nodes $= q^{O(\log k)}$.

Each node yields a cost that is polynomial in q.

Total cost: $q^{O(\log q)}$.

Embedding in larger fields

Let $\mathbb{F}_{p^n}$, with $p < n$, which is not of the **appropriate shape**.

- If $p \approx n$, then set $q = p$ and $k = n$, and embed the DLP in $\mathbb{F}_{q^{2k}}$.
  The input size is **twice as large**: preserve the complexity.
- If $p = 2$ (worst case), set q to the smallest power of 2 larger than n, and set $k = n$.
  We work in a field of order $2^{2n \log n}$.
  The complexity w.r.t. to original size is also **preserved**!
- If $p \gg n$, the complexity is no longer controlled.

Quasi-polynomial algorithm

Main result (based on heuristics)

Let K be a finite field of the form $\mathbb{F}_{q^k}$. A discrete logarithm in K can be computed in heuristic time

$$\max(q, k)^{O(\log k)}.$$

Cases:

- $K = \mathbb{F}_{2^n}$, with prime n. Complexity is $n^{O(\log n)}$. Much better than $L_{2^n}(1/3 + o(1)) \approx 2^{\sqrt[3]{n}}$.
- $K = \mathbb{F}_{q^k}$, with $q \approx k$. Complexity is $\log Q^{O(\log \log Q)}$, where $Q = \#K$. Again, this is $L_Q(o(1))$.
- $K = \mathbb{F}_{q^k}$, with $q \approx L_{q^k}(\alpha)$. Complexity is $L_{q^k}(\alpha + o(1))$, i.e. better than Joux-Lercier or FFS for $\alpha < 1/3$.

Plan

Presentation of the problems

Combining congruences

DLP in finite fields of small characteristic

- The BaGaJoTh quasi-polynomial DLP algo
- Practice: $L(1/4)$ algo by Joux

The linear algebra step of the number field sieve

Limitation of BaGaJoTh

The main **drawback** of the quasi-polynomial is

$$\boxed{\text{The arity of the descent tree is} \approx q^2.}$$

Joux's $L(1/4)$ algorithm:

- **Arity** only $\approx q$;
- Tree is **deeper**.

Let's concentrate on the **bottom of the tree**, where polynomial systems are involved.

Setting

Same context as before:

$$\mathbb{F}_{q^{2k}}, \quad \text{with} \quad q \approx k.$$

Goal: for P of degree $D < \sqrt{k}$, rewrite $\log P$ in terms of logs of polys of smaller degree.

Using again the key equation

Let k_1 and k_2 be two polynomials; and plug k_1/k_2 in the field equation $X^q - X = \prod(X - \alpha)$:

$$k_1(X)^q k_2(X) - k_1(X)k_2(X)^q = \prod_{(\alpha:\beta)\in\mathbb{P}^1(\mathbb{F}_q)} \beta k_1(X) - \alpha k_2(X).$$

Wishlist on k_1 and k_2 we are looking for:

- k_1 and k_2 have **small degree** (hence RHS is smooth);
- P **divides** the LHS;
- the rest of the LHS is **smooth**;

Modeling divisibility by a bilinear system

Idea: $x \mapsto x^q$ is a high degree function, but is **linear**. Make this q-linearity explicit.

Step 1: Write an explicit $\mathbb{F}_q$-basis for $\mathbb{F}_{q^2}$:

$$\mathbb{F}_{q^2} = \mathbb{F}_q[t]/(t^2 + \tau_1 t + \tau_0).$$

Step 2: Fix degrees for k_1 and k_2 and write $\mathbb{F}_q$-indeterminates for their coeffs:

$$\begin{aligned} k_1(X) &= a_0 + a_1 X + \cdots + a_{d_1} X^{d_1} \\ &= (a'_0 + a''_0 t) + (a'_1 + a''_1 t)X + \cdots + (a'_{d_1} + a''_{d_1} t)X^{d_1}, \\ k_2(X) &= b_0 + b_1 X + \cdots + b_{d_2} X^{d_2} \\ &= (b'_0 + b''_0 t) + (b'_1 + b''_1 t)X + \cdots + (b'_{d_2} + b''_{d_2} t)X^{d_2}. \end{aligned}$$

Step 3: Expand / collect expressions like $k_1(X)^q k_2(X)$:
Get a polynomial in X and t with coefs that are **bilinear**.

Modeling divisibility by a bilinear system (2)

Step 4: Do this for the whole LHS and write the generic remainder modulo P:
This is a polynomial of degree $D-1$ in X, degree ≤ 1 in t, and each coeff is a bilinear form in the $\mathbb{F}_q$-indeterminates:

$$\{a_i, a_i''\} \quad \text{and} \quad \{b_j', b_j''\}.$$

Step 5: Write that the remainder is zero:
This gives $2D$ bilinear equations over $\mathbb{F}_q$.

Degrees of freedom: taking into account a few redundancies, we need

$$d_1 + d_2 = D + 1.$$

Algorithm: Write the system, solve it, test smoothness of the LHS.

Complexity analysis

Naïve approach: take $d_1 \approx d_2 \approx D/2$, and use classical complexity estimates for Gröbner basis computation.

FAIL: exponential cost.

Recent result of Faugère, Safey El Din, Spaenlehauer:

Theorem on general bilinear system solving

For a 0-dimensional affine bilinear system involving n_x and n_y unknowns, the solutions can be found in time

$$O\left(\binom{n_x + n_y + \min(n_x + 1, n_y + 1)}{\min(n_x + 1, n_y + 1)}^{\omega}\right).$$

Conclusion: it is better to unbalance d_1 and d_2.

Complexity analysis (2)

Small or large d_1, d_2?:

- If the degrees are very unbalanced, the depth of the tree becomes too large (one step make the degree of P diminish slowly);
- If the degree are not too much unbalanced, the cost of the polynomial system resolution becomes too large.

Final complexity:
If we start with a polynomial P of degree $D = \sqrt{q}$, then the best choice is

$$d_1 \approx \sqrt[4]{q}.$$

The global cost of the bottom of the descent tree is then

$$\boxed{L_{q^{2k}}(1/4 + o(1)).}$$

Practical?

Record of Granger, Kleinjung, Zumbrägel (2014): DLP in

$$\mathbb{F}_{2^{9234}} \equiv \mathbb{F}_{2^{18\times 513}}.$$

Algorithms:

- Top of the tree: used a "classical" descent (à la FFS).
- From degree 9 to degree 2: descent with bilinear systems.
- Logs of degree 1 and 2: same as in quasi-polynomial algo.

Costs of Gröbner basis computation with Magma:

degree	time
≤ 7	seconds
8	45 minutes
9	5 hours

Only 1% of the total time (45 years on 1 core).

Plan

Presentation of the problems
- Cryptographic background
- Primality, smoothness, factorization
- The discrete logarithm problem

Combining congruences
- Basic subexponential factoring algorithm
- Combining congruences for DLP

DLP in finite fields of small characteristic
- The BaGaJoTh quasi-polynomial DLP algo
- Practice: $L(1/4)$ algo by Joux

The linear algebra step of the number field sieve
- Overview of the NFS algorithm
- Linear algebra: the filtering step

Plan

Presentation of the problems

Combining congruences

DLP in finite fields of small characteristic

The linear algebra step of the number field sieve

- Overview of the NFS algorithm
- Linear algebra: the filtering step

Setting

Back to prime fields!

Let p be a (large) prime.
Goal: solve DLP in $\mathbb{F}_p$.

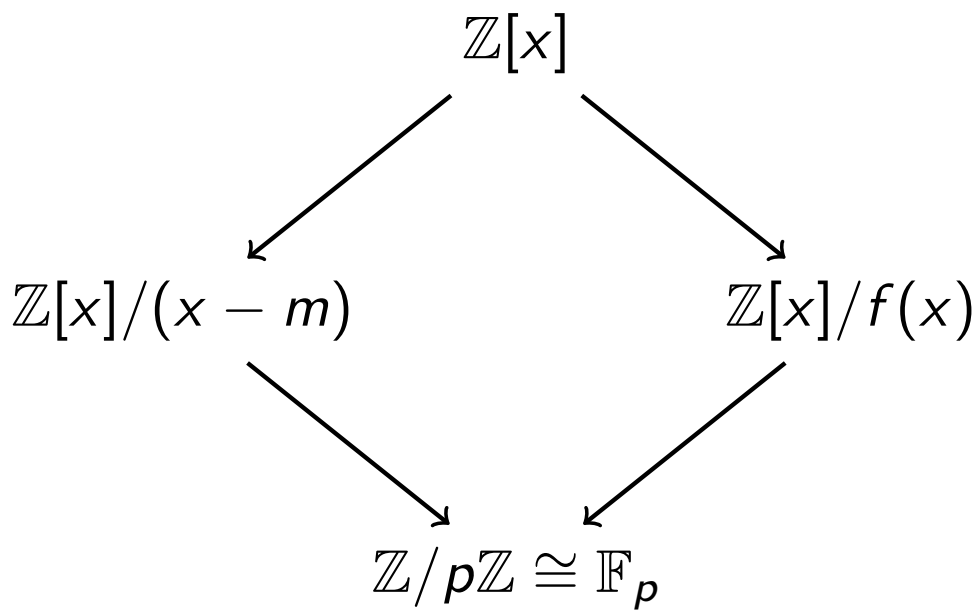
Remarks:

- By Pohlig-Hellman, work in a subgroup of prime order $\ell \mid p-1$.
- p (and ℓ) are **multi-precision** primes: if they fit in 64 bits, then BSGS takes less than 2^{32} operations (a few seconds on one core).
- Current record: p with 180 digits $\approx$ 600 bits.

The magic diagram of NFS

Let $f(x)$ be a polynomial and m an integer such that $f(m) \equiv 0 \mod p$. We denote by α the algebraic number that is a root of f.

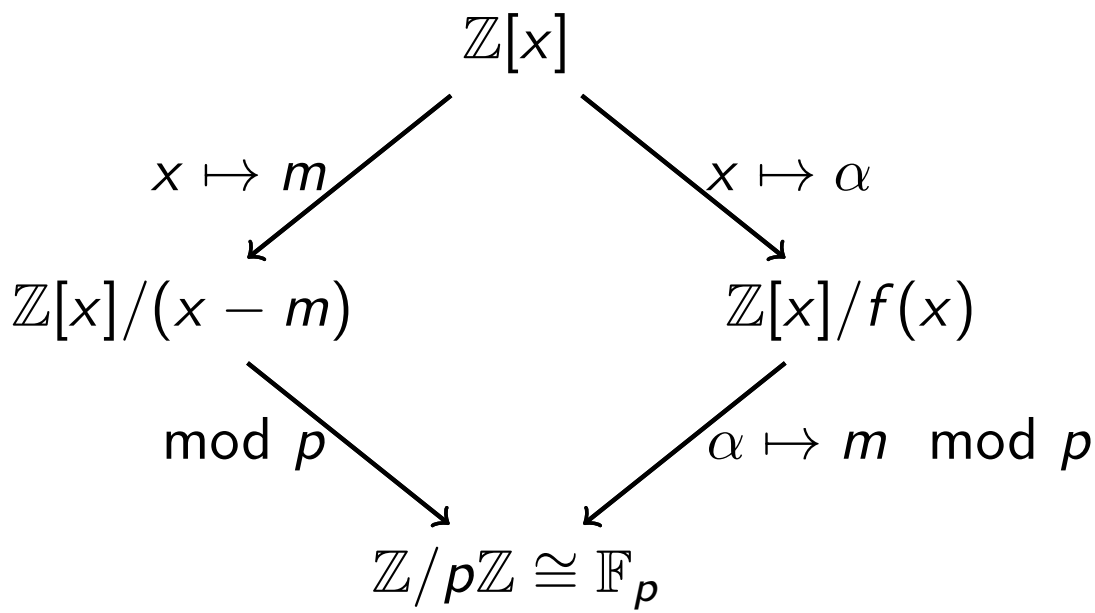
The **diagram commutes** (the maps are **ring** homomorphisms).



The magic diagram of NFS

Let $f(x)$ be a polynomial and m an integer such that $f(m) \equiv 0 \mod p$. We denote by α the algebraic number that is a root of f.

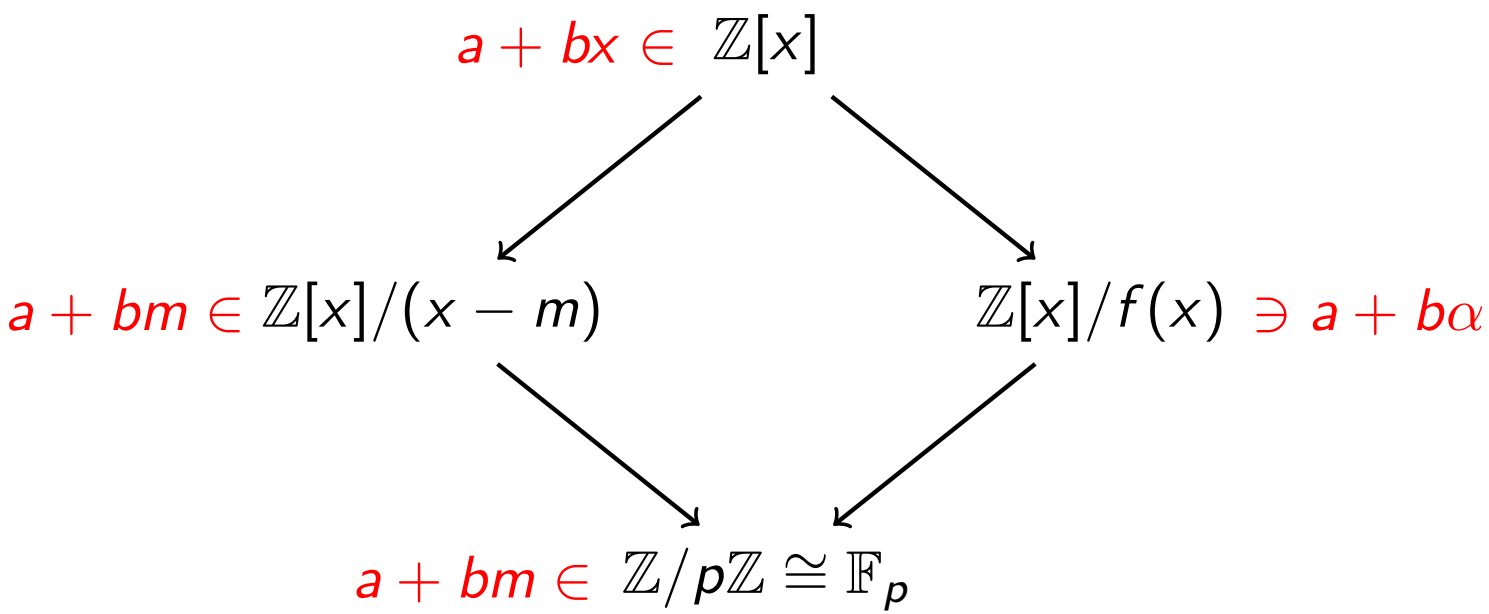
The **diagram commutes** (the maps are **ring** homomorphisms).



The magic diagram of NFS

Let $f(x)$ be a polynomial and m an integer such that $f(m) \equiv 0 \mod p$. We denote by α the algebraic number that is a root of f.

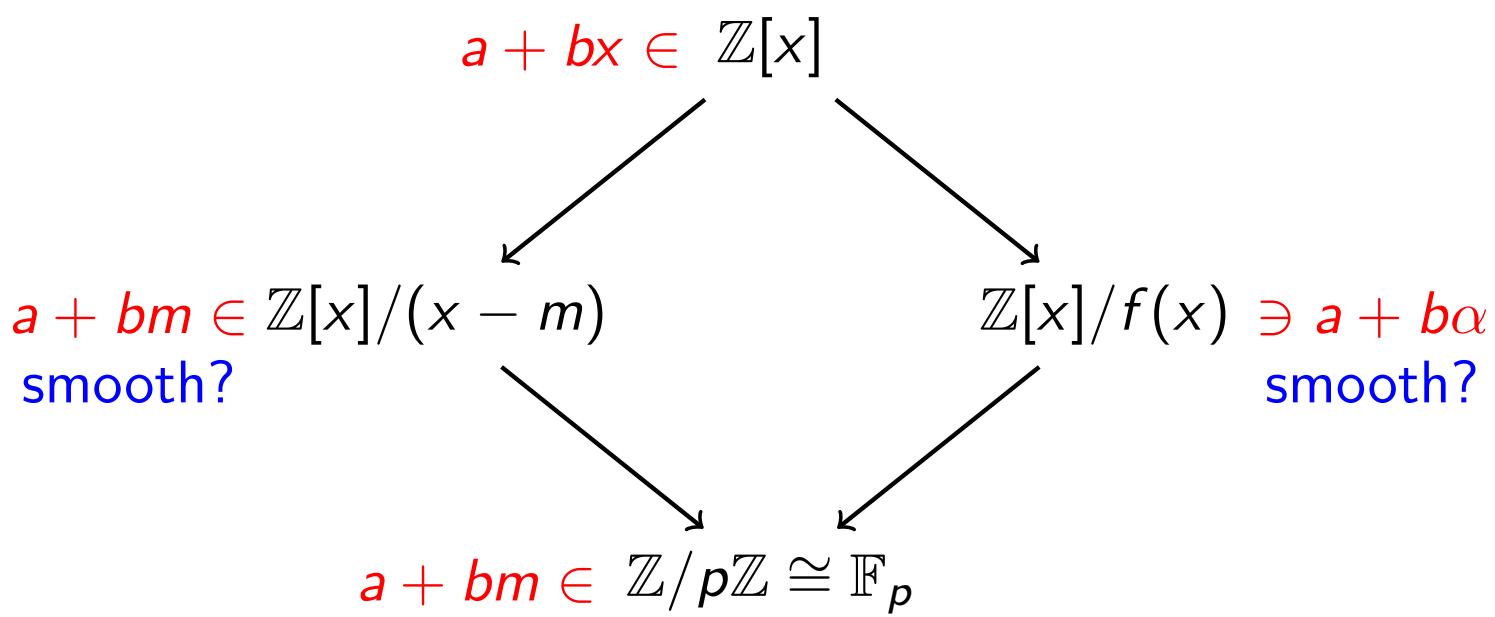
The **diagram commutes** (the maps are **ring** homomorphisms).



The magic diagram of NFS

Let $f(x)$ be a polynomial and m an integer such that $f(m) \equiv 0 \mod p$. We denote by α the algebraic number that is a root of f.

The **diagram commutes** (the maps are **ring** homomorphisms).



If smooth on both sides, then we get a **relation** in $\mathbb{F}_p$.

What does it mean to be smooth?

On the **rational side** (left): smoothness of integers. OK.

On the **algebraic side** (right):

- Smoothness in a number ring.
- In general, this is not a Unique Factorization Domain.
- Have to factor **ideals**.
  An ideal is smooth iff its norm is smooth.
- A lot of (theoretical and practical) technicalities to define the "log of an ideal mapped to $\mathbb{F}_p$."
  Work of Schirokauer.

Rem. Main mathematical structure: **Dedekind domain**.
Algorithms for manipulating ideals have been developped in the late 80's (see Cohen's book).

General algorithm

1. **Select** f and g appropriate for given p;
2. Try many candidates for $a - bx$ and check whether they are **smooth** on both sides;
3. Transform them into **relations**:

$$\begin{aligned}(a - b\alpha) &= \prod_{\mathrm{N}(\mathfrak{q})<B} \mathfrak{q}^{e_{\mathfrak{q}}} \\ a - bm &= \prod_{q<B} q^{e_q}.\end{aligned}$$

4. Add the Schirokauer maps to **convert to linear equations** between logs.
5. Solve the **linear algebra** system modulo ℓ.
6. Use a descent procedure to rewrite the **target** element into small elements of known logs.

A few words on Schirokauer maps (SM)

To the plain sparse matrix, we need to **add SMs columns** to take into account units.

Theorem of Dirichlet

Let f be a polynomial over $\mathbb{Z}$. Let r_1 be the number of real roots and $2r_2$ the number of complex roots.
The rank of the unit group of the number field of f is $r_1 + r_2 - 1$.

Facts:

- The **number of SMs** columns to add is $r_1 + r_2 - 1$.
- These columns are **dense** in both sense:
  - No zero entries;
  - Entries are random-looking elements modulo ℓ.

Rem. The rest of the matrix is sparse in both sense: few entries, most of them ± 1.

Complexity analysis (sketch)

Fix $d = \sqrt[3]{3 \log p / \log\log p}$.

Let $m \approx p^{1/d}$, and f given by the **base-m expansion** of p:

$$p = f_0 + f_1 m + \cdots + f_{d-1} m^{d-1} + m^d = f(m),$$

where $f_i \approx m \approx p^{1/d} = L_p(2/3)$.

Taking a and b of size $\approx L_p(1/3)$, we get objects to smooth of size:

$$L_p(2/3).$$

Setting a **smoothness bound** $B = L_p(1/3)$, we get a probability of smoothness on each side of $L_p(1/3)^{-1}$.
Sparse **linear algebra**: $\tilde{O}(B^2) = L_p(1/3)$.

[ugly details skipped...] Overall complexity in

$$\boxed{L_p(1/3, \sqrt[3]{64/9} + o(1)).}$$

Complexity analysis – comments

What gives the gain $L_p(1/2)$ to $L_p(1/3)$?

- In basic combination of congruences: one object to smooth, of size $L_p(1)$.
- In NFS: two objects to smooth of size $L_p(2/3)$.
- Splitting in **two paths** was the key.

Going further?

- Would need to split in **more than two pieces** to test for smoothness.
- Drawing a diagram with objects of **dimension more than 2** seems like a good idea (didn't manage to make it work...)
- Any idea, someone???

Plan

Presentation of the problems

Combining congruences

DLP in finite fields of small characteristic

The linear algebra step of the number field sieve

- Overview of the NFS algorithm
- Linear algebra: the filtering step

A running example: p180

Computation finished in June 2014: Bouvier, G., Imbert, Jeljeli and Thomé.

Input:

- p of 180 digits, random-looking: $p = \texttt{RSA180} + 625942$;
- $p - 1 = 2\ell$, with ℓ prime;
- target: log of a random-looking element: $\texttt{RSA1024}$.

Data on the computation:

- Polynomial selection: **60 days**.
- Relation collection: 253 M relations in **50 years**.
- Filtering: reduce to a matrix of size 7.2 M with 150 entries per row.
- SM: **1 year**.
- Linear algebra with Wiedemann: **80 years**.
- Individual logs: a few hours.

A few words on the Wiedemann part

Iterative method: main workhorse is Sparse Matrix $\times$ Vector.

Usually, two levels of **parallelism**:

- Use of **block** Wiedemann: several independant sequences in parallel.
- Each sequence run the SpMv on a set of interconnected, multicore nodes:
  - maybe the matrix does not fit in a single node;
  - parallelism without overhead in the Berlekamp-Massey step.

Main specific property: SM columns should be handled in a different way.

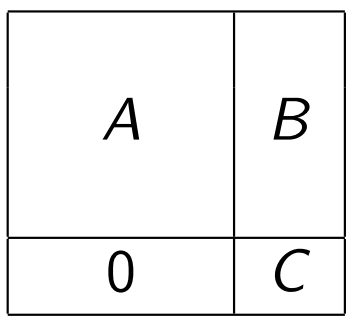
Example of `p180`**.** Blocking factor: $(12, 24)$. Each sequence on 4 nodes with 16 core each at 2.7 sec per iteration.
Berlekamp-Massey step took 15 hours on 144 cores.

Filtering step: generalities

Remember the p180 example:

- Matrix ouput by the relation collection: 253M rows, 82M columns, 20 non-zero entries per row.
- Matrix entering Wiedemann: 7.2M rows/cols, 150 non-zero entries per row.

More precisely:
We look for a non-zero **right-kernel** vector (length = 82M).
From a non-zero kernel vector of the small matrix (length = 7.2M), we can deduce the others.



A is in row echelon form.

Properties of input matrix

Atypical properties: (non-SM part)

- Very **sparse**: only a few non-zero entries per row (say 20). Variance is reasonably low (no row with 3 or 100 entries).
- Most entries are 1's.
- The **column density** goes from close to one (column labelled by prime 2) to almost or completely empty.
- No structural property.
- There are **rows that are equal** (!)
  This is due to the relation collection algorithm.
- There are empty column.
- A lot of **redundancy**: we collect much more rows than required for having maximal rank.

Trivial things are not trivial anymore

Memory issue:

- On p180, the raw relations take 20 GB of (compressed) space on disk.
- If we add the SMs, this would add 50 GB: postpone their computation as much as possible.

Duplicate removal:

Trivial mathematically; needs to be done on disk.
Example p180: keep 175M rows from the 253M initial ones.

Singleton removal:

Again trivial mathematically:

- Detect and remove empty columns.
- Pivot with column that have only 1 non-zero entry.

Strategy for "removing columns": first renumbering of columns after duplicate removal, second renumbering just before Wiedemann.

Using redundant rows

Entering this stage:

A_2	B_2
0	C_2

A_2 is almost Id.

On p180: C_2 has 171M rows, 78M cols. All columns of C_2 have at least 2 non-zero entries.

Clique removal step: select rows of C_2 to be deleted to transform it into a square matrix.
Graph-theoretic point of view:

- Each node is a row;
- If a column has weight 2, put an edge between the two corresponding rows.

Using redundant rows (2)

Select a large **connected component** (was called "clique" by factorization people!!!):

- Remove one of the row of this CC;
- This creates chained singleton columns, so that the whole CC can be "removed".

Number of CC that can be removed:

$$\#\mathrm{rows}(C_2) - \#\mathrm{cols}(C_2).$$

Heuristics:

- Start with largest CC.
- Can refine the quality of the CC by counting the weights of the corresponding rows.

Example `p180`: reduce to a square matrix of size 21M.

Let's start real pivoting

Up to now, no real pivoting: rows have not been combined yet.

The final step of the filtering is usually called **merge**.

Goal: select pivots so that matrix remains as sparse as possible.

Strategy:

- Select columns of smallest weight.
- Eliminate the corresponding rows.
- Repeat until the matrix is too dense.

Pivoting of a column is optimized with a **minimum spanning tree** computation.
Stopping criterion: based on an **estimate of the cost** of the Wiedemann step.

Example p180: final matrix size is 7.2 M with 150 non-zero entries per row.

Filtering, summary

Summary of the filtering steps:

- Duplicate removal;
- Singleton removal;
- Clique removal;
- Merge.

Rem. All of this takes more or less **linear time**, as opposed to quadratic for Wiedemann.
In practice, can try several set of parameters, and choose the best result for Wiedemann (based on bench, not estimates).

Recent idea of Kleinjung (for factorization): store the final matrix as a product of larger matrices, so that the total weight is smaller.

Conclusion

Integer factorization and **discrete logarithm** involve:

- Analytic number theory (for the complexity estimates);
- Algebraic number theory (ugly details of NFS);
- Arithmetic geometry (elliptic curves for ECM);
- Computer arithmetic (fast implementation at low-level);
- Parallelism (fast implementation at high-level);
- **Computer algebra**:
  - Asymptotically fast algorithms;
  - Exact linear algebra;
  - Polynomial systems;

In each domain, can take "off the shelf" tools; but looking under the hood almost systematically helps.

References

Books:

- *Mathematics of Public Key Cryptography* by Steven Galbraith (CUP, 2012).
- *Prime Numbers: A Computational Perspective* by Richard Crandall and Carl Pomerance (Springer, 2001).
- *Algorithmic cryptanalysis* by Antoine Joux (CRC, 2009).
- *The Development of the Number Field Sieve* by Arjen Lenstra and Hendrik Lenstra (Springer, 1993).

Software:

- CADO-NFS. `http://cado-nfs.gforge.inria.fr`
- GMP-ECM. `http://ecm.gforge.inria.fr`
- LinBox. `http://www.linalg.org`