Arithmetic algorithms for cryptology — 2 February 2015, Paris

Linear algebra and NFS

Razvan Barbulescu
CNRS and IMJ-PRG

Outline of the talk

- Linear algebra
- The number field sieve

Linear algebra in cryptography

- what to solve? DLP and factoring modern algorithms, of complexity $L(1/3)$ or less require to solve $Mw = 0$, for a sparse matrix M (few non-zero entries per row).
- $K = ?$ For factoring $K = \mathbb{F}_2$,
  while for DLP $K = \mathbb{F}_\ell$ with $|\ell|$ between 200 and 600 bits
- software
  - very few implementations available for linear algebra software over so large fields, e.g. LINBOX uses fields of 32 bits
  - CADO: linear algebra over $\mathbb{F}_2$; Hamza Jeljeli: software for $\mathbb{F}_p$
- extra difficulty For some DLP algorithms we have a few (in practice 1 to 6) heavy columns which must be treated separately and require large amounts of memory.

Full vs sparse linear algebra[1]

Full matrix algorithms

- space $O(N^2)$, matrix represented in memory as arrays
- time $O(N^\omega)$, same as matrix multiplication, where
  - Gauss (naive multiplication) $\omega = 3$;
  - Strassen $\omega = 2.81$;
  - Coppersmith-Winograd $\omega = 2.38$;
  - open problem: $\omega = 2$?

Sparse matrix algorithms

- space $N\lambda$,
  - matrix stored as a list of $N\lambda$ pairs $(i, j_{i,0})$, ..., (i, j_{i,k_i}) containing the positions of non-zero entries;
  - matrix is read-only, we implement matrix times vector multiplication and use it as a black box (building block)
- time $O(N^2\lambda)$, representing $O(N)$ calls of the black box.

[1] Courtesy to E. Thomé

Wiedemann algorithm: main idea

Problem

Given a field K and a matrix $M \in \mathrm{Mat}_{N\times N}(K)$, with λ non-zero entries per row, such that $\det M = 0$. In $O(N^2\lambda)$ operations over K, find a non-zero solution of

$$Mw = 0.$$

Sketch of the solution

1. Find a polynomial h in $K[x]$ such that $h(M) = 0$ (for example the characteristic polynomial).
2. Since $\det M = 0$, $h(x) = xh^{-}(x)$ for a polynomial h^{-}.
3. Pick a random vector u and evaluate $w = h^{-}(M)u$.
4. We have: $Mw = Mh^{-}(M)u = h(M)u = 0u = 0$.
5. If h was chosen of minimal degree then $h^{-}(M) \neq 0$. Since u is random we will show that, with high probability, $w \neq 0$.

Retrieving the linear generator

The problem

Given a sequence generated by a linear recurrence, find the linear generator:

- for $1, 10, 100, 1000, \ldots$, find $1 - 10x$;
- for $1, 1, 2, 3, 5, 8, 13, 21, \ldots$, find $1 - x - x^2$.

Formalization

Let $a_0, a_1, \ldots$ be a sequence given by the recurrence

$$\forall k,\ a_k = -\lambda_1 a_{k-1} - \lambda_2 a_{k-2} - \cdots - \lambda_n a_{k-n}.$$

Then, the linear generator $\Lambda = 1 + \lambda_1 x + \cdots + \lambda_n x^n$ satisfies

$$(a_0 + a_1 x + \cdots a_{2n-1} x^{2n-1})\Lambda(x) \equiv (b_0 + \cdots + b_{n-1} x^{n-1}) \mod x^{2n},$$

for some scalars $b_0, \ldots, b_{n-1}$.

Solutions

- $(s_0, r_0) = (0, x^{2n})$ and $(s_1, r_1) = (1, \sum_{i=0}^{2n-1} a_i x^i)$
- if (s, r) and (s', r') are solutions, any combination $(\alpha(x)s + \beta(x)s', \alpha(x)r + \beta(x)r')$ is also solution, $\alpha, \beta \in K[x]$.

Extended Euclid algorithm (EEA)

Algorithm

Input two polynomials $f, g \in K[x]$ with $\deg f, \deg g \leq 2n$ and $\deg(\gcd(f,g)) < n$.

Output a sequence of triples $(r_i, t_i, s_i) \in K[x]^3$ such that $r_i = t_i f + s_i g$, and $\deg r_0 \geq \deg r_1 \geq \ldots \geq \deg r_k$, where $r_k = \gcd(f,g)$

$\begin{pmatrix} r_1 & t_1 & s_1 \\ r_0 & t_0 & s_0 \end{pmatrix} \leftarrow \begin{pmatrix} g & 0 & 1 \\ f & 1 & 0 \end{pmatrix}$

$i \leftarrow 1$

while $\deg r_i \geq n$ **do**

  $q_i \leftarrow r_{i-1} \operatorname{div} r_i$

  $\begin{pmatrix} r_{i+1} & t_{i+1} & s_{i+1} \\ r_i & t_i & s_i \end{pmatrix} \leftarrow \begin{pmatrix} -q_i & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} r_i & t_i & s_i \\ r_{i-1} & t_{i-1} & s_{i-1} \end{pmatrix}$

  $i \leftarrow i + 1$

end while

return $\{(r_0, t_0, s_0), \ldots, (r_i, t_i, s_i)\}$

Properties

- The sequence $(\deg r_i)$ is strictly decreasing, while $(\deg t_i)$ and $(\deg s_i)$ increasing.
- For all i, $\deg s_i = \deg q_0 + \cdots + \deg q_{i-1} = \deg r_0 - \deg r_{i-1}$.

Complexity for $\deg f, \deg g \leq n$

EEA costs $O(n^2)$, fast algorithms cost $O(M(n)(\log n))$.

Computing linear generators with EEA

Theorem

EEA applied to $f = x^{2n}$ and $g = \sum_{k=0}^{2n-1} a_k x^k$ outputs (r, t, s) such that s is a linear generator.

Proof.

- $r_0 = x^{2n}$ and $r_1 = \sum_{k=0}^{2n-1} a_k x^k$ have a GCD of degree less than n because the sequence (a_k) is not identical zero. So $\deg r_i < n$ for large enough i.
- Let i_0 be the last value of i in the algorithm, i.e.

  $$\deg r_{i_0} < n \leq \deg r_{i_0-1}.$$

  We have $\deg s_{i_0} = \deg r_0 - \deg r_{i_0-1} = 2n - \deg r_{i-1} \leq n$.
- The pair (s, r) satisfies the definition of the linear generator.

□

Berlekamp-Massey (alternative to compute linear generator)

- comes from the theory of error correcting codes (BCH);
- complexity $O(n^2)$, same as EEA;
- fast variants of complexity $O(M(n)(\log n))$ (same as EEA);
- unlike EEA, generalizes to linear generators over matrices (Thomé 2003, etc).

Wiedemann

Algorithm

Input An $N \times N$ singular matrix M over a field K

Output a non-trivial solution of $Mu = 0$

```
x←Random(K^{N×1}), y←Random(K^{1×N}),
[Krylov] Compute a_i = y M^i x for i in [0, 2N − 1]
[Linear generator] Compute the linear generator Λ = Σ_{i=0}^{deg Λ} c_{deg Λ−i} x^i of (a_n)_{n∈ℕ}
h(x) ← Σ_{i=0}^{deg Λ} c_i x^i
h^−(x) ← x^{−val_x h} h(x)
v ←Random(K^{N×1});                              ▷ can be v ← x
[Make solution] u ← h^−(M) v
repeat
    u ← Mu
until  Mu = 0 and u ≠ 0                          ▷ ≤ N + 1 − deg Λ times
```

Complexity

- one product matrix times vector costs $N\lambda$ multiplications in K
- Krylov: $2N^2\lambda + N^2$ operations in K
- Linear generator: N^2 (complexity of EEA). But there exist fast algorithms of complexity $O(N(\log N)^2)$.
- make solution: $N^2\lambda$ (Horner algorithm)
- total: $(3 + o(1))N^2\lambda$.

Correctness

Notation

- μ=minimal degree monic polynomial such that $\mu(M) = 0$;
- μ_x=minimal degree monic polynomial such that $\mu_x(M)x = 0$;
- $\mu_{x,y}$=minimal degree monic polynomial such that $y^t\mu_{x,y}(M)x = 0$.

Theorem

If x and y are randomly chosen in a finite field K, then, with probability greater than or equal to $1 - \min(1, O(\frac{N}{\#K}))$,

$$\mu = \mu_x = \mu_{x,y}.$$

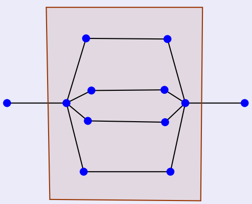
Proof.

- Clearly $\mu_{x,y}$ divides μ_x, which divides μ.
- Given a polynomial μ', $\mu'(M)x = 0$ if and only if x is in a vectorial subspace, so it occurs with probability $O(1/\#K)$. Since μ has at most N cofactors of irreducible divisors, the failure probability is $\min(1, O(N/\#K))$
- Given a vector x and a polynomial μ' such that $\mu'(M)x \neq 0$, we have $y^t\mu'(M)x \neq 0$ except if we picked by error y in the hyperplane of vectors perpendicular on $\mu'(M)x$. This occurs with probability $O(1/\#K)$.

□

Two levels of parallelism

Inside the black box

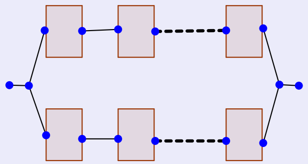


⋆ matrix product is done block-wise

⋆ each computing unit (node or CPU cores/pair of GPUs) stores one block of the matrix, so we parallelize the memory

⋆ synchronize after each call to the block box: high speed communication (InfiniBand or two GPU's on the same PC)

Outside the block box



⋆ unlike Wiedemann, there exists an algorithm where one has independent sequences of computations

⋆ no communication at all (for this level of parallelism)

⋆ each sequence stores the complete matrix in memory: no memory parallelism

⋆ compatible with the parallelism inside black box

Other algorithms

Lanczos

- comes from the EDP context;
- does Gram-Schimdt orthogonalization w.r.t M^tM;
- cost $(2+o(1))N^2\lambda$.

Block Wiedemann

- Krylov and Make solution allow outside black box parallelism;
- minimal polynomial with coefficients in $m \times n$ matrices, e.g. 2×2;
- linear generator is slowed down by a small factor depending on m and n;
- when $K = \mathbb{F}_2$ and $n = 32$, 32 calls of the black box cost as much as one call when $K = \mathbb{F}_\ell$ when ℓ has 32 bits.

Block-Lanczos

If computations are done on n cores, we synchronize after each iteration, so N/n synchronizations. This is better than Wiedemann, not as good as Block Wiedemann.

In practice

- DLP: all algorithms can be used;
- factoring: Lanczos and Wiedemann might fail

Non-homogeneous systems and left-hand kernel

Left kernel

Sparse matrices are stored as a list $\{(0, i_{0,0}), \ldots, (1, i_{1,0}), \ldots\}$. We can sort the list on the second coordinate, in quasi-linear time $O(N\lambda)$ and obtain the sparse representation of M^t. Then we apply Wiedemann to M^t.

Solve $Mw = b$ 1st method

Apply the homogeneous algorithm to the system

$$\left(\begin{array}{ccc|c} & M & & b^t \\ \hline 0 & \cdots & 0 & 0 \end{array}\right)(x|x_{N+1})^t = 0.$$

Rescale the solution so that the last coordinate is -1.

Non-homogeneous systems: 2nd method

Solve $Mw = b$, 2nd method

Wiedemann is modified as follows:

1. Compute a polynomial h so that $h(M)b = 0$.
2. Write $h = xh^{-}(x) + h_0$ and compute $w = \frac{-1}{h_0} h^{-}(M)b$.

Block Wiedemann can also be modified.

Getting rid of heavy columns (Thomé, to be published)

- In the 2014 record computation of DLP in $\mathbb{F}_p$, the computations related to 2 heavy columns took half of the time of linear algebra (homogeneous system).
- E. Thomé proposed to use the Block Wiedemann with the heavy columns as vectors b, thus solving a non-homogeneous system.

Outline of the talk

- Linear algebra
- The number field sieve

The benefit of commutative diagrams

Example for DLP (with Gaussian integers)

- Goal: DLP in $\mathbb{F}_p$ for $p \equiv 1 \mod 4$.
- Compute a root of $r^2+1=0$ in $\mathbb{F}_p$ and put $f=x-r$ and $g=x^2+1$.
- Compute pairs of integers (a,b) such that $F(a,b)=a-rb$ and $G(a,b)=a^2+b^2$ are smooth.
- Factor $a-br=\prod q_i^{e_i}$ and $(a-\sqrt{-1}b)=\prod(\pi_j+\sigma_j\sqrt{-1})^{\epsilon_j}$ ($\mathbb{Z}[\sqrt{-1}]$ is an unique factorization ring).
  Since $G(a,b)=a^2+b^2=\prod_j(\pi_j^2+\sigma_j^2)$, all q_i, π_j and σ_j are small.
- We obtain in $\mathbb{F}_p^*$:

$$\prod q_i^{e_i} \equiv a-br \equiv \prod(\pi_j+\sigma_j r)^{\epsilon_j} \mod p.$$

- Take discrete log to obtain a linear equation.
- Continue as in Index Calculus.

What changed?

If f and g have small coeffs, we replace the smoothness probability of a large integer by two tiny integers simultaneously.

Polynomial selection

Goal

Find two polynomials f and g with a common root modulo a given integer (composite N for factoring or prime p for DLP).

Gaussian integers

In the previous example we can use rational reconstruction(EEA) to write $r \equiv u/v \mod p$ with $u, v \approx \sqrt{p}$. Replace f by $u - xv$, so $\|f\|_\infty \approx \sqrt{p}$. Then

1. $F(a,b) \approx \sqrt{p}$,
2. $G(a,b)$ tiny.

Is as if we tested smoothness for numbers of size $\sqrt{p}$ instead of p.

Base-m

Put $m = \lfloor N^{1/d} \rfloor$ and write $N = m^d + N_{d-1}m^{d-1} + \cdots N_1 m + N_0$ in base M and put

- $f = x^d + \cdots + N_1 x + N_0$;
- $g = x - m$.

We have $|F(a,b)| \approx E^d m$ and $|G(a,b)| \approx Em$ where E upper bounds $|a|$ and $|b|$.

Change of complexity: $L(1/3)$

Tiny quantities?

We have $|F(a,b)| \approx E^d m$ and $|G(a,b)| \approx Em$ where E upper bounds $|a|$ and $|b|$.

Goal

The key fact in the DLP algorithms is the size of the smoothness bound and of the quantities which must be smooth. How to choose everything small?

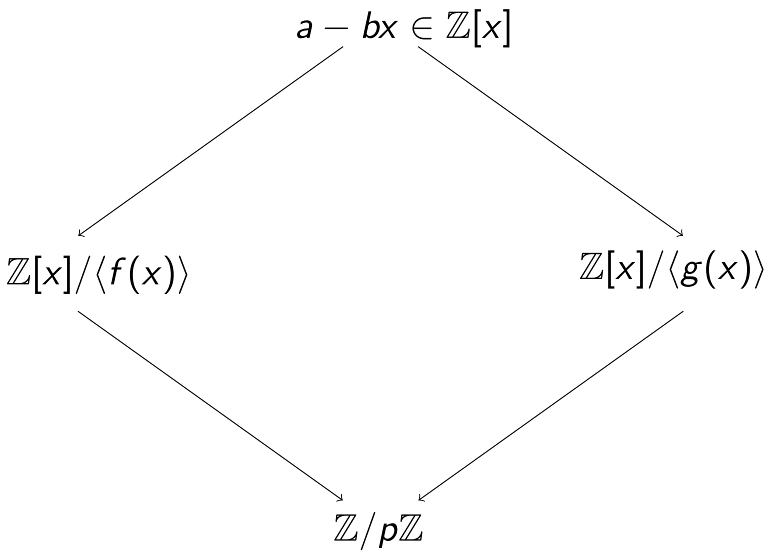
Order of magnitude

- Write the smoothness bound $B = L(1/3)$, the bound on the integers (a, b), $E = L(1/3)$ and $d = (\log N / \log\log N)^{1/3}$. Then
  - $F(a,b) = L(1/3)^d L(1)^{1/d} = L(2/3)$;
  - $G(a,b) = L(1/3) L(1)^{1/d} = L(2/3)$;
- smoothness probability $= 1/L(2/3 - 1/3) = 1/L(1/3)$ so we need $L(1/3)$ pairs a, b.
- OK because $E = L(1/3)$.

The number field sieve (NFS): diagram

NFS for DLP in $\mathbb{F}_p$

Let $f, g \in \mathbb{Z}[x]$ be two irreducible polynomials, which have a common root m modulo p.



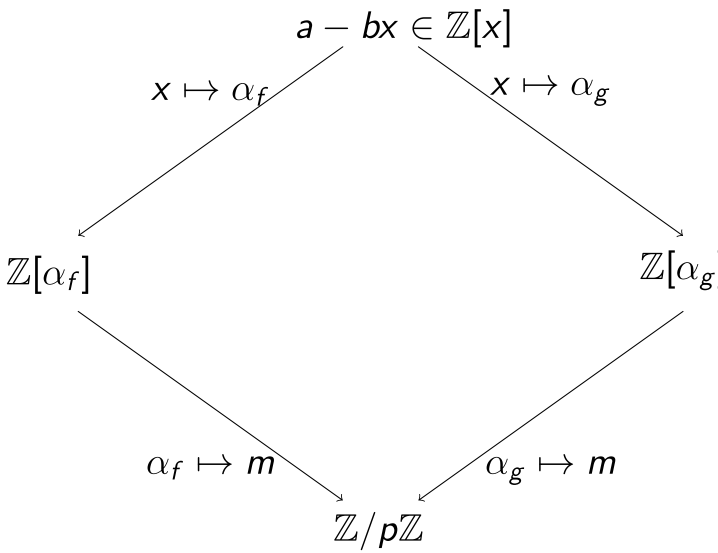
Computations in $\mathbb{Z}[\alpha_f]$?

- Mathematical parts of the code are negligible, albeit they produce bugs.
- Implementations available: PARI/GP, Magma, CADO.

The number field sieve (NFS): diagram

NFS for DLP in $\mathbb{F}_p$

Let $f, g \in \mathbb{Z}[x]$ be two irreducible polynomials, which have a common root m modulo p.



Computations in $\mathbb{Z}[\alpha_f]$?

- Mathematical parts of the code are negligible, albeit they produce bugs.
- Implementations available: PARI/GP, Magma, CADO.

NFS: algorithm for DLP

Input a finite field $\mathbb{F}_{p^n}$, two elements t (generator) and s
Output $\log_t s$

1: (Polynomial selection) Choose two polynomials f and g in $\mathbb{Z}[x]$ such that one has the diagram presented before;

2: (Sieve) Collect coprime pairs (a, b) such that $F(a, b)$ and $G(a, b)$ are B-smooth (for a parameter B);

3: Write a linear equation for each pair (a, b) found in the Sieve stage.

4: (Linear algebra) Solve the linear system to find (virtual) logarithms of the prime ideals of norm less than B;

5: (Individual logarithm) Write $\log_t s$ in terms of the previously computed logs.

Factor base

Here we factor into prime ideals of the two number fields. We have $F(a, b)G(a, b)$ smooth if and only if $(a - \alpha_f b)$ and $(a - b\alpha_g)$ factor into ideals of the factor base.

NFS: algorithm for factorization

Input an integer N
Output with probability 50% a non-trivial factor of N

1: (Polynomial selection) Choose two polynomials f and g in $\mathbb{Z}[x]$ such that one has the diagram presented before;

2: (Sieve) Collect coprime pairs (a, b) such that $F(a, b)$ and $G(a, b)$ are B-smooth (for a parameter B);

3: Write an exponent vector for each pair (a, b) found in the Sieve stage, modulo 2.

4: (Linear algebra) Find a linear combination of the rows of M which sum to zero;

5: (Square root) Compute a product in the number fields to obtain $X^2 \equiv Y^2 \mod N$.

Success probability

Using Block Wiedemann we compute 32 or more solutions at a time. We only repeat the Square root stage. We succeed with probability $1 - 2^{-32}$.

NFS: sieve

Naive variant (1989)

1. For f, enumerate integers a and , for each, sieve the polynomial $F(a,x)$; obtain pairs (a,b) where $F(a,b)$ is smooth.
2. For g, do the same to find pairs (a,b) where $G(a,b)$ is smooth.
3. Intersect the two sets.

Special-Q (1993)

Given a prime q, and a root r such that $f(r) \equiv 0 \mod q$ we compute two rational reconstructions (EEA)

$$r \equiv \frac{a_0}{b_0} \equiv \frac{a_1}{b_1} \mod q,$$

with $a_0, b_0, a_1, b_1 \approx \sqrt{q}$. Then we sieve the pairs (i,j) so that

- $F(a_0 i + a_1 j, b_0 i + b_1 j)/q$ is B-smooth;
- $G(a_0 i + a_1 j, b_0 i + b_1 j)$ is B-smooth.

advantage With almost no extra work, we know that we sieve the pairs which have at list one factor q which is large, but smaller than the smoothness bound.

Franke-Kleinjung (2009)

We enumerate directly the vectors of a lattice.

Individual logarithm

Smoothing (also called continued fraction descent)

When computing $\log_t s$, as in Index calculus, we test random i until $t^i s \mod p$ is B-smooth.

If $P(x, y)$ is the probability that a number less than x is y-smooth, then one can prove that

$$P(x_1, y)P(x_2, y) \geq P(x_1 x_2, y).$$

Hence, we do a rational reconstruction (EEA) of ($t^i s$ mod p) before testing smoothness.

Descent by special-Q

We fix $0 < c < 1$. If the log of g is required, we search a pair (a, b) so that $G(a, b)/q$ is a q^c-smooth integer and $F(a, b)$ is q^c-smooth. Hence we obtain a relation between $\log q$ and logs of smaller ideals.

In short, individual log consists of:

1. The smoothing stage allows to write the log of a number of size $L(1)$ as $(\log q)^{O(1)}$ logs of primes of ideals of size $L(2/3)$.
2. By the descent stage, one writes the logs of size $L(2/3)$ as $\log q^{O(\log q)}$ primes and ideals of size $L(1/3)$, in the factor base.

Conclusion

- Cryptographic algorithms require sparse algebra
  - $\mathbb{F}_2$ for factoring;
  - $\mathbb{F}_\ell$ with large ℓ for DLP;
  - parallelism is a problem, although Block-Wiedemann allows perfect parallelism for ≤ 10 computing sites.
- NFS is the best algorithm for factoring and the best algorithm for DLP in $\mathbb{F}_p$ (and large characteristic).
- NFS has complexity $L(1/3)$ because it requires smoothness for numbers of size $L(2/3)$.
- NFS was improved especially by accelerating the sieve and making linear algebra parallel.