Random problems at the core of integer factorization algorithms

Pierrick Gaudry

Caramba – LORIA, Nancy
CNRS, Université de Lorraine, Inria

Journées ALEA, March 2016

Plan

Introduction: crypto context

Integer factorization 101

How to quickly test smoothness?

A random structure in the test

Conclusion

A few words of crypto

Public key cryptography was invented in the 70's.
This solves major practical problems for the deployment of crypto in everyday life:

- Key exchange over an insecure channel;
- Certificates (be sure that you are talking to the right person);
- Signatures.
- ...

The **RSA** algorithm is still widely used. Security relies on the presumed difficulty of **integer factorization**.

$$n = p\,q$$

Example: EMV

EMV is the standard for chip-and-PIN **payment cards**.

- Widely used in Europe (and in France, with Carte Bleue);
- A 20-year old standard;
- Use *vintage* crypto algorithms: Triple-DES, SHA-1, RSA;
- RSA with key up to 1984 bits hard-coded in the standard.

What's in my card?

Example: EMV

EMV is the standard for chip-and-PIN **payment cards**.

- Widely used in Europe (and in France, with Carte Bleue);
- A 20-year old standard;
- Use *vintage* crypto algorithms: Triple-DES, SHA-1, RSA;
- RSA with key up to 1984 bits hard-coded in the standard.

What's in my card?

Script: courtesy of E. Thomé and J. Detrey, based on standard Linux tools.

Plan

Introduction: crypto context

Integer factorization 101

How to quickly test smoothness?

A random structure in the test

Conclusion

Integer factorization is hard

Some integers are **easy to factor**:

- prime numbers (cf Bill Gates, Millenium 4, ...);
- prime powers;
- **smooth** numbers: all their prime factors are small;
- (smooth number) $\times$ (prime);
- $p \times \texttt{nextprime}(p)$;
- ...

In general (and for RSA numbers), best (heuristic) complexity is

$$\exp\left(1.902\ (\log n)^{1/3}(\log\log n)^{2/3}\right).$$

Worse than polynomial, but better than exponential.

Integer factorization is hard

Best complexity for integer factorization:

$$\exp\left(1.902\ (\log n)^{1/3}(\log\log n)^{2/3}\right).$$

Def. Security parameter = log of time of the attack.

Here:

$$\boxed{\text{Security} \approx \sqrt[3]{\text{key size}}}$$

For comparison:

- For an **ideal system**, the best attack would be exhaustive search: security $\approx$ key size.
- For a **disastrous system**, the best attack takes polynomial time: security $\approx \log$(key size).

Fermat: difference of squares

If one finds two integers x and y such that

$$n = x^2 - y^2,$$

then n can (maybe) be factored as $n = (x - y)(x + y)$.

More generally, one can look for x and y such that

$$x^2 \equiv y^2 \mod n,$$

and $x \not\equiv \pm y \mod n$.

Rem. This is the basis of the **quadratic sieve** and of the **number field sieve** leading to the complexity above.

Combining congruences

Let's pick a **random** x modulo n.
Compute $z \equiv x^2 \mod n$ as an integer in $[0, n-1]$.
The chances that z is a square y^2 are **exponentially small**.

Smoothness to the rescue!

Def An integer z is B-smooth if all its prime factors are $< B$.

Find many x_i's such that result is B-smooth:

$$\begin{aligned} x_0^2 &\equiv 2^{e_{0,0}}\ 3^{e_{0,1}}\ \cdots\ p_k^{e_{0,k}} \mod n \\ x_1^2 &\equiv 2^{e_{1,0}}\ 3^{e_{1,1}}\ \cdots\ p_k^{e_{1,k}} \mod n \\ x_2^2 &\equiv 2^{e_{2,0}}\ 3^{e_{2,1}}\ \cdots\ p_k^{e_{2,k}} \mod n \\ \vdots &\qquad\qquad \vdots \end{aligned}$$

Goal: Multiply together a subset of these relations to get a square on the RHS.

Combining congruences – 2

Write the exponents in a matrix, one row per relation:

$$M = \begin{pmatrix} e_{0,0} & e_{0,1} & \cdots & e_{0,k} \\ e_{1,0} & e_{1,1} & \cdots & e_{1,k} \\ e_{2,0} & e_{2,1} & \cdots & e_{2,k} \\ \vdots & & \vdots & \end{pmatrix}$$

Find a non-zero **vector** v in the **left-kernel** of M:

$$v\ M = 0.$$

Rem. Only parity of the exponents is relevant: do this computation in $\mathbb{F}_2$.

Then v tells which relations to combine to **get a square**:

$$\prod_{i \text{ s.t. } v_i=1} x_i^2 \equiv \square \mod n$$

How frequent are smooth numbers?

If smoothness bound B is very large: very frequent.
If B is tiny: very rare.

Choose B in between: what we need!

Def. $\psi(x, y)$: number of y-smooth integers smaller than x.

Thm. (CEP, 1983) Let $u = \frac{\log x}{\log y}$. We have:

$$\Psi(x, y)/x = \exp(-u(\log u + \log\log u - 1 + o(1))),$$

assuming u not too close to 1. The $o(1)$ is under control.

Rule of thumb: take Probability(smooth) $\approx \rho(u) \approx u^{-u}$.

Tuning the smoothness bound

In the previous algorithm, the **optimal bound** is of the form

$$B = \exp(c\sqrt{\log n}\sqrt{\log\log n}).$$

The **total cost** is then of the same form (with another c).

The exact **constants** in the exponent depends on

- How to test for smoothness?
  - Trial division;
  - Sieving;
  - Elliptic curves.
- How to do the linear algebra?
  - Gauss: cubic time;
  - Strassen, . . . ;
  - Iterative methods (sparse matrix): quadratic time.

Lowering the complexity: NFS

The **number field sieve** (NFS):

- Invented by Pollard, Lenstra, Lenstra, ...; early 90's.
- Use number fields and a smoothness notion for **ideals**.
- Main feature: reduces the **size** of the integers to test for smoothness from $\approx n$ to $\approx \exp((\log n)^{2/3})$.
- In practice, starts to win around 100 digits.

The general idea stays the same: combining congruences.

Plan

Introduction: crypto context

Integer factorization 101

How to quickly test smoothness?

A random structure in the test

Conclusion

Main task in the NFS algorithm

Two tasks takes almost all the time in NFS:

- **Collect relations**:
  Find many pairs of coprime integers (a, b) such that $f(a, b)$ and $g(a, b)$ are simultaneously smooth for some fixed polynomials f and g.

- **Linear algebra**:
  Find a non-zero left-kernel vector of a matrix over $\mathbb{F}_2$.

Latest record RSA-768 (done in 2010) provides some data.

Sieving

$$f(a,b) = f_6 a^6 + f_5 a^5 b + f_4 a^4 b^2 + \cdots + f_0 b^6,$$

Just like in Eratosthenes:

$$\boxed{\text{If } p | f(a,b), \text{ then } p | f(a+kp, b+k'p).}$$

Strategy:

- Look for (a,b) in a box $[-I, I[\times[0,J[$;
- Initialize a 2-dim array;
- Loop over all prime $p < B$:
  - Find a first position where p divides;
  - Visit all the other positions (boing, boing!);
  - Remember which ones are divisible by p.
- Collect results.

Problem. Not enough memory!

Sieving as a prefilter

A **filtering strategy:**

1. **Sieve** with a bound $B' < B$;
2. **Discard** (a, b)-pairs which don't look promising;
3. For the survivors, use (batch'd) trial division or ECM to **finish** the smoothness test.

ECM = elliptic curve method (Lenstra 85).

A **survivor** that enters step 3 looks like (m not too large):

$$f(a,b) = \underbrace{p_1 \times p_2 \times \cdots \times p_k}_{B'\text{-smooth part}} \times \underbrace{m}_{B\text{-smooth?}}$$

ECM for testing smoothness – 1

ECM is a **probabilistic algorithm** that extract prime factors.

Main **building block**:

- take an integer m as input;
- choose a parameter B_1;
- choose a (random) **elliptic curve** E over $\mathbb{Q}$;
- do some computation in E modulo $\mathbb{Z}/m\mathbb{Z}$;

Features:

- the **runtime** is roughly proportional to B_1;
- maybe the algorithm returns a **proper factor** of m;
- in nothing is returned, the **probability** that there is a prime factor of b bits in m can be **bounded**;
- the bound depends only on B_1 and b.

ECM for testing smoothness – 2

Now, iterate the process with **many curves** and tune B_1.

We obtain **a Las Vegas algorithm** that

- takes an integer m as input;
- takes a target prime size B as a parameter;
- after a time $O(\exp(\sqrt{2 \log B \log\log B})(\log m)^2)$:
  - maybe returns a proper factor of m;
  - If nothing is returned, the probability that m has a prime factor p less than B is $< 1/2$.

Rem. Probability of failure can be made arbitrarily small, but you will not know for sure that the input m is not smooth.

Rem. The parameter B_1 and the number of curves to try grow like $\exp(\sqrt{\frac{1}{2} \log B \log\log B})$. (e.g. $B_1 = 500$, number of curves $= 20$)

Plan

Introduction: crypto context

Integer factorization 101

How to quickly test smoothness?

A random structure in the test

Conclusion

Which criterion for being a survivor?

$$f(a,b) = \underbrace{p_1 \times p_2 \times \cdots \times p_k}_{B'\text{-smooth part}} \times \underbrace{m}_{B\text{-smooth?}}$$

Which criterion for being a survivor?

$$f(a,b) = \underbrace{p_1 \times p_2 \times \cdots \times p_k}_{B'\text{-smooth part}} \times \underbrace{m}_{B\text{-smooth?}}$$

- If $m < B'$: bug in the sieve!
- If m is prime: can readily decide.
- If $m > B'^2$ and $m < B^2 < B'^3$: can have only 2 prime factors. The more ECM fails to factor m, the more likely it is B-smooth!
  Here, strategy is clear: continue until we factor completely the number.

But: For current and future records, we need to allow 3, 4 or maybe more prime factors in m.
[Why? Asymptotically, this depends on your computational model: Turing machine, circuit...]

Basic early abort criteria

m: remaining unfactored part
B': sieve bound; no prime $< B'$ in m
B: smoothness bound

Fact. Let $k \geq 2$, and assume that

$$m \in [B^k, B'^{k+1}],$$

then m can not be B-smooth.

> *If $m > B^k$ is B-smooth, it has at least $k+1$ prime factors.*
> *Since all of them are $> B'$, then m must be $> B'^{k+1}$.*

Rem. For large k the interval is empty and the statement is void.

Let's draw a picture...

Turning these into probabilities

In the remaining intervals, it is possible to compute the probability that m is B-smooth, assuming it is not prime.

- Heuristic: m is uniformly random among non-prime numbers with given size, without prime factors $< B'$;
- Play Lego with primes and count how many are B-smooth. *variants of* $\Psi()$ *function; want approximate values, not asymptotic.*

Consequence. If probability is small, just discard the survivor wiithout testing it with ECM.

Refining the probabilities during ECM

Trying one curve in ECM has an associated succes probability depending on the size of the target prime factor.

- Can be estimated asymptotically;
- Can be computed for the actual implementation by sampling.

Consequence. After each failed trial, it is possible to **update the knowledge** on the probability of B-smoothness of m.

Rem. In the Shadok case, the success probabilities increase!

Varying ECM parameters

ECM can be tuned to target **any smoothness bound**, not necessarily the bound B which is the final target.
Of course, a larger smoothness bound means a costly ECM step.

Strategy: starts with **fast ECMs** targeting primes smaller than B:

- Maybe there is one in m.
  Could in principle estimate the probability that this is the case, and update it after each ECM failure!
- Even if there is not, this might be a way to **gain knowledge** (update probabilities) faster than with a more costly ECM.
- Possibly have a better choice for the subsequent expensive ones.

Let's summarize

Input: the unfactored part m, with no primes $< B'$.

We want to find a **sequence of ECMs parameters** that:

- Finds quickly a factor $< B$ of m if there is one;
- Accumulates quickly knowledge so that we can discard it if it is not B-smooth.

How many curves in the sequence? A few dozens, at most.
How much time per survivor? A few milliseconds, at most.

In practice: Kleinjung's approach

Main (only?) **reference on the topic**:
T. Kleinjung. *Cofactorization strategies [...]*, 2006.

- **Heuristic**: extracting the first prime factor is the dominant part. *Becoming more and more wrong...*
- Some **genetic algorithm** to pre-select good chains of ECMs.
- **Convexity arguments** for choosing the best chain for each size of m.

Much better in practice than a naive approach: was used for the latest record.

On the theoretical side

There are two contexts where tuning an ECM chain improved the **asymptotical complexity**.

- Bernstein-Lange (in Batch NFS, 2014): essentially same context as here.
  Only interested in asymptotics; circuit model (no sieve!).
  Improved the second term in the runtime estimates.
- Barbulescu (in PhD, 2013): discrete logarithm context, final step.
  Asymptotics; classical model.
  Improved the exponent.

These works are of essentially no use in practice except for general guidance.

Wanted! Something in between

General **feeling** that there must be a better answer to the problem:

- Pure asymptotic complexities are too far away from reality;
- Kleinjung's approach might miss a lot: too many heuristics;
- Could AofA give an answer? Something between
  - Analytic number theory;
  - Operational Research.

First step: need a good modeling...

Even some insight about what's going on in Kleinjung's approach would be useful.

Remember: In the end, there is an implementation to optimize!

Plan

Introduction: crypto context

Integer factorization 101

How to quickly test smoothness?

A random structure in the test

Conclusion

Conclusion

- A better understanding of some random process could lead to important speed-up in integer factorization algorithms.
- Most of it applies as well to discrete logarithm in prime fields.
- A related question:
  - Predict accurately the properties of the matrix;
  - Deduce the time for the linear algebra step;
  - Generate random matrices with these properties to test software *before* running the record!