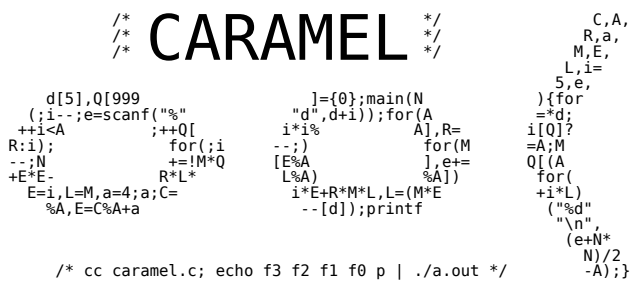
Square Root Algorithms for the Number Field Sieve

E. Thomé[1]

[1] INRIA/CARAMEL, Nancy.



Jul. 19th, 2012

Plan

Why integer factorization ?

($N = pq \rightarrow$ find p, q) is a hard problem.

An old yet very modern question.

- Comes back to antiquity.
- Associated to big names like Gauss, Fermat.
- Became a crucial question for cryptography in the 70's, with RSA.
- A variety of algorithms, some obsolete, some still in use.
- Best candidate for attacking RSA is the Number Field Sieve, invented in the early 90's.

Record sizes: crypto in sight

The feasibility limit explored by factoring records is used to determine key sizes for RSA.

- SSL/TLS. CA root certificates are installed by default in browsers.
  - Linux laptop, 2005: 1024b (50%), 2048b (48%), 4096b (2%) ;
  - Linux laptop, 2009: 1024b (31%), 2048b (58%), 4096b (10%).
  - Linux laptop, 2012: 1024b (16%), 2048b (66%), 4096b (16%).

- EMV credit cards (a.k.a. chip and pin).

  Most chip public keys are 960b. Some 1024b (until end of 2009, some had a 896b key).

Factoring experiments: decision-driving data for setting key sizes.

The number field sieve (NFS)

The number field sieve (NFS) is the "king" of factorization algorithms:

- NFS has the best asymptotic complexity.
- NFS merges many difficulties:
  - close to impossible to prove its complexity;
  - difficult to predict its practical running time;
  - the implementation can be endlessly optimized.
- NFS contains many other integer factorization algorithms as subroutines:
  - Trial division;
  - Sieving (à la Eratosthenes);
  - $p-1$, $p+1$, the Elliptic Curve Method (ECM);
  - (optionally) MPQS, SQUFOF, Rho.

Focus points of this talk

- NFS description
- Square root algorithms
  - "Naive" (but fast enough) algorithm.
  - More algorithms.

CADO-NFS: an implementation of NFS

Implementing NFS is a significant effort.
(NFS is much more involved than e.g. ECM).

Our experimentation playground: CADO-NFS.

- New implementation started afresh in 2007.
- Actively developed. Playground for new ideas.
- Certainly beatable, but contains nice algorithms.
  - State-of-the art or close to it.
  - Largest factored general number with CADO-NFS: RSA-704. (to be compared to RSA-768)
- No refrain to reorganizing the code to (changing) taste every so often.
- written (almost) entirely in `C`. As of 2012, $\sim$ 150 kLOC.
- LGPL, `http://cado-nfs.gforge.inria.fr/`

Plan

Introduction

Description of NFS

Square root algorithms

The GNFS setup

For factoring "general" N, GNFS uses:

- a number field $K = \mathbb{Q}(\alpha)$ defined by $f(\alpha) = 0$, for f irreducible over $\mathbb{Q}$ and $\deg f = d$;
- Another irreducible polynomial g such that f and g have a common root m mod N (example: $g = x - m$).

g defines the rational side, f defines the algebraic side.

Choosing f and g is referred to as the polynomial selection step.

General plan: Obtain relations, and combine them to obtain:

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

Relations in NFS

$$\begin{array}{ccccc} & & \mathbb{Z}[x] & & \\ & \psi^{(1)} : x \to m \swarrow & & \searrow \psi^{(2)} : x \to \alpha & \\ & \mathbb{Z}[m] & & \mathbb{Z}[\alpha] & \\ \varphi^{(1)} : t \to t \mod N \searrow & & & \swarrow \varphi^{(2)} : \alpha \to m \mod N & \\ & & \mathbb{Z}/N\mathbb{Z} & & \end{array}$$

Take for example $a - bx$ in $\mathbb{Z}[x]$. Suppose for a moment that:

- the integer $a - bm$ is smooth: product of factor base primes;
- the algebraic integer $a - b\alpha$ is also a product.

Then we have an multiplicative relation in $\mathbb{Z}/N\mathbb{Z}$. We can hope to combine many such relations to form a congruence of squares.

$$R = (a_1 - b_1 m) \times \cdots \times (a_k - b_k m) = \square,$$
$$A = (a_1 - b_1 \alpha) \times \cdots \times (a_k - b_k \alpha) = \square,$$
$$\varphi^{(1)}(R) \equiv \varphi^{(2)}(R) \mod N.$$

Recognizing when $a - b\alpha$ factors

Obstruction: $\mathbb{Z}[\alpha]$ not a UFD. "Factoring" $(a - b\alpha)$ won't work.

Better: hope for

- the integer $a - bm$ being smooth
  (product of factor base primes);
- the ideal $(a - b\alpha)$ being smooth as well
  (product of factor base prime ideals).

Some obstructions must be dealt with or worked around:

- ramifications,
- who's the maximal order.

Computationally (for $(a - b\alpha)$), we want the integer

$$\text{Norm}_{K/\mathbb{Q}}(a - b\alpha) = \text{Res}(a - bx, f) = b^d f(a/b) = F(a, b)$$

to be smooth. Nothing terribly complicated.

Complexity of NFS

For factoring an integer N, GNFS takes time:

$$L_N[1/3, (64/9)^{1/3}] = \exp\left((1 + o(1))(64/9)^{1/3}(\log N)^{1/3}(\log\log N)^{2/3}\right).$$

This is sub-exponential.

Note: some special numbers allow for a faster variant NFS, with complexity

$$L_N[1/3, (32/9)^{1/3}] = \exp\left((1 + o(1))(32/9)^{1/3}(\log N)^{1/3}(\log\log N)^{2/3}\right).$$

NFS essentials

NFS might not be the simplest algorithm on earth, but:

- obstructions have been dealt with already long ago. See literature.
- the bottom line is simple: everything boils down to assembly/C/MPI.

Polynomial selection: find f, g;

Sieving: find many a, b s.t. $F(a, b) = b^d f(a/b)$ and $G(a, b)$ smooth.

Linear algebra: combine a, b pairs to get a congruence of squares. ($\Rightarrow$ solve a large sparse linear system over $\mathbb{F}_2$.)

Square root: complete the factorization.

Recent progresses

Since RSA-155 (512 bits) in 1999, many improvements.

- Much better polynomial selection (Kleinjung, 2003, 2006).
- Very efficient sieving code (Franke, Kleinjung, 2003–).
- Very efficient cofactorization code (Kleinjung, Kruppa).

More recent state of the art, notably for linear algebra:

- Use block Wiedemann algorithm (BW), at separate locations.
- Use computer grids idle time to do linear algebra.
- Use sequences of unbalanced length in BW.

All contributed to the factorization of RSA-768 (2010).

Computational load

RSA-768: largest general number factored ever.

Work by Lausanne, Nancy, Bonn, Tokyo, Amsterdam, ...

- Polynomial selection:
  - 40-core years.
  - Settled on one polynomial.
- Sieving:
  - $\approx$1500 core-years.
  - Used idle time on many clusters.
  - Collected 64 billion relations, 5.4TB
- Linear algebra:
  - 100GB matrix, 192M$\times$ 192M, 144 nz/row.
  - Block Wiedemann algorithm.
    Used up to 8 clusters simultaneously.
  - $\approx$ 200 core-years, but expensive hardware.
- Square root ? Much, much less.

Plan

Introduction

Description of NFS

Square root algorithms

Forcing squares

Recall where relations are coming from:

- the integer $a_i - b_i m$ is smooth. $\quad a_i - b_i m = \prod p_j^{e_j}$,
- the ideal $(a_i - b_i \alpha)$ is also smooth. $\quad (a_i - b_i \alpha) = \prod \mathfrak{p}_k^{e'_k}$,

We consider a combination $S(x) = \prod_i (a_i - b_i x)$.

- How do we make $S(m)$ a square in $\mathbb{Q}$?
- How do we make $S(\alpha)$ a square in $\mathbb{Z}[\alpha]$?

(we focus on the algebraic side.)

Building squares: Characters

First part of the answer: linear algebra.

- By solving a linear system, we can force all valuations ot be even.
- Is this enough ? Clearly not.

We only guarantee that $\mathfrak{I} = (S(\alpha))\mathcal{O}_K$ is the square of an ideal.

- $\mathfrak{I}$ need not be the square of a principal ideal ;
- $S(\alpha)$ need not be a square in $\mathbb{Q}(\alpha)$;
- $S(\alpha)$ need not be a square in $\mathbb{Z}[\alpha]$.

We can overcome these obstructions heuristically with characters.

Input of the square root step

- Linear algebra $\Rightarrow$ all valuation even.
- Characters step $\Rightarrow \prod_i (a_i - b_i\alpha) = \square$ in $\mathbb{Z}[\alpha]$.

Thus

$$(a_1 - b_1 m) \times \cdots \times (a_s - b_s m) \equiv \phi((a_1 - b_1\alpha) \times \cdots \times (a_s - b_s\alpha)) \mod N$$

is actually a congruence of squares

BUT computing the square root is in non trivial, esp. on algebraic side.

Square root psychology

The sqrt step is
- mathematically interesting
- rarely an issue computationally speaking.

However it comes LAST in a long computation.

- The "user" is in front of his terminal.
- Having to wait is annoying, esp. if code improvements are easily obtained.

The algebraic square root:

- gathers most mathematical difficulties.
- is computationally harder.

$\Rightarrow$ Focus on alg. sqrt.

Approaches for sqrt

We investigate several approaches.

- All algorithms are linear or quasi-linear in the input size.
- However the input is large. 21GB for RSA-768.
  Input known as a product form $\prod(a_i - b_i\alpha) = S(\alpha)$.
- The output really is $T(m) \bmod N$ where $T(\alpha)^2 = S(\alpha)$.
  If possible, try to avoid computing $T(\alpha)$ itself.

The different approaches compute different things.

- Some compute $S(\alpha)$, some don't.
- Some compute $T(\alpha)$, some compute $T(m) \bmod N$ directly.
- Some exploit the known factorization of $(a - b\alpha)$.

NFS square root 20 years ago

May rephrase the problem as factoring $X^2 - S(\alpha)$ in $K = \mathbb{Q}(\alpha)$.

- Quasi-linear (we take $K = \mathbb{Q}(\alpha)$ constant).
- BUT unacceptably expensive 20 years ago.
- Motivation to explore specially adapted algorithms:
  - Montgomery;
  - Couveignes.

NFS square root 20 years ago

May rephrase the problem as factoring $X^2 - S(\alpha)$ in $K = \mathbb{Q}(\alpha)$.

- Quasi-linear (we take $K = \mathbb{Q}(\alpha)$ constant).
- BUT unacceptably expensive 20 years ago.
- Motivation to explore specially adapted algorithms:
  - Montgomery;
  - Couveignes.

Since then: practicality of asymptotically fast methods everywhere.

- May explore again the direct approach.
  - Easy to program.
  - Can leverage fast implementations in e.g. Gnu MP.
- We are also interested in parallelization.

The best way

The most efficient algorithm for the square root is Montgomery's.

Montgomery's algorithm much different from the ones described in the talk.

- Requires some number-theoretic primitives not there in typical NFS code.
  - Zassenhaus round-2.
  - Complete ideal factorization.
  - Arbitrary ideal arithmetic.
- We know no LGPL-licensed code we can directly reuse.
- Not really an excuse, but still cumbersome enough to explore the other algorithms.

Plan

Square root algorithms

- The lifting approach
- Couveignes' algorithm
- Another CRT method

The direct (lifting) approach

The direct/lifting/naive/brute-force way:

- Compute $S(\alpha)$ from the set of $a_i - b_i\alpha$.
- Compute $T(\alpha)$ as a square root in $\mathbb{Z}[\alpha]$.
- Deduce $T(m)$ mod N.

Key: take an inert prime p: such that $p\mathcal{O}_K$ is prime.

- The field $K_p = K \otimes \mathbb{Z}_p$ is a field extension of $\mathbb{Q}_p$.
- Elements of $\mathbb{Z}[\alpha]$ easy to recognize in K_p: expansion terminates.
- p-adic structure allows for a lifting approach.

Lifting approach: strategy

- Projection map $\pi : \mathbb{Z}_p[\alpha] \to \mathbb{F}_{p^d}$ to the residue field.
- Consider $t \in K_p$ an arbitrary lift of $\pm\sqrt{\pi(S(\alpha))}$.
  We thus have $t - T(\alpha) \equiv 0 \bmod p$.
- $T(\alpha)$ is a fixed point of $x \to x + \frac{S(\alpha)-x^2}{2x}$.
- lift, lift, lift.

Requirements
- Arithmetic in K_p (fixed precision will do).
- Fast integer multiplication.
- How high should we lift ?
- Inert primes...

$T(\alpha)$ determined only up to sign !

There is no such thing as "the" square root.

The choice of t determines the $T(\alpha)$ obtained by lifting.

How high is the lift ?

Rough estimate:
- Coefficients of $S(\alpha)$ have, say, n bits.
- So coefficients of $T(\alpha)$ should have $\approx n/2$ bits.

Easy to make it slightly more serious.

- Compute FP approximations to $\log|a_i - b_i z|$.
  - z runs over complex roots of f (recall $f(\alpha) = 0$).
  - Round towards $+\infty$.
- We obtain an upper bound for $\log|T(z)|$.
- Derive upper bound on coefficients of T.
- Can restrict the lift to sufficiently large p^λ.

Inert primes ?

- (Čebotarev) The density of primes inert in a degree d number field is proportional to the ratio

$$\frac{\#\{\sigma \in G,\ \mathrm{ord}(\sigma) = d\}}{\#G}$$

  where G is the Galois group.
- But that number could be zero ! E.g. for $G \cong \mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z}$.
- Fortunately for GNFS, polynomials have the generic $G = \mathfrak{S}_d$. Not so for SNFS.
- The lifting approach does not work as is for non-generic Galois group.

Lifting approach implementations

For RSA-768, the lifting approach could not handle the sqrt step.

- Memory requirement probably $\approx$ 100GB. We did not have that much RAM by then (2010).
- No longer an issue now. Presumably will not be an issue for next big factorization either.

Several existing implementations (msieve, cado-nfs).

Without inert primes

It is possible to do without inert primes.
(e.g. Pari does that to factor polynomials in number fields.)

- Pick a prime which is "as inert as it can be": splits in the smallest possible number of factors.
  (This may still mean up to $d/2$ factors.)
- Compute desired roots modulo each of the several factors of $p\mathcal{O}_K$, and lift appropriately.
- Find the correct combination.

This roadmap will be explored later in this talk.

Plan

Square root algorithms

- The lifting approach
- Couveignes' algorithm
- Another CRT method

Working modulo many primes ?

1990 era:
- no widely available FFT implementation for multiplying integers (GMP has one only since 1999).
- Explore ways to avoid it.

Can't we work with several primes ?

Suppose we have 100 inert primes.

- Compute square root modulo $p_0, \ldots, p_{99}$.
- Try to recombine via CRT.
- Problem: only 2 correct square roots among 2^{100} possible combinations !

We need a way to tell apart the two square roots consistently mod p.

Couveignes' algorithm

Couveignes' trick for **odd-degree number fields**

We have $\mathrm{Norm}_{K/\mathbb{Q}}(-T(\alpha)) = -\mathrm{Norm}_{K/\mathbb{Q}}(T(\alpha))$.
Thus for $t \equiv \pm T(\alpha) \bmod p$, we have:

$$\mathrm{Norm}_{\mathbb{F}_{p^d}/\mathbb{F}_p}(t) = \pm \mathrm{Norm}_{K/\mathbb{Q}}(T(\alpha)) \bmod p.$$

- $\left|\mathrm{Norm}_{K/\mathbb{Q}}(T(\alpha))\right|$ is something we can compute.
- Define the square root $T(\alpha)$ as the one with positive norm.
- This leads to a well-defined choice of root modulo each p.

Notations
- Let $\{p_i\}$ be a collection of sufficiently many primes.
- Let $T_i(x)$ be such that $T_i(\alpha) \equiv T(\alpha) \bmod p_i$.
- Let $q_i \in [0, \prod_i p_i[$ satisfy $q_i \bmod p_j = \delta_{i,j}$.

Couveignes' algorithm

We have $T(\alpha) = \sum_i q_i T_i(\alpha)$.

- We need not compute $T(\alpha)$.
  Directly computing $T(m)$ mod N is simpler.
  $\Rightarrow$ accumulate the contributions modulo each p_i.

Complexity:

- dominated by the computation of $\{S(\alpha) \bmod p_i\}$.
- if $S(\alpha)$ is read for each p_i, quadratic complexity.

Key characteristics:

- Limited to odd degree.
- Requires inert primes.

Plan

Square root algorithms

- The lifting approach
- Couveignes' algorithm
- Another CRT method

Another CRT method

Goal: have something which works in the CRT way, but:

- without requiring inert primes.
- with good complexity.

We only target a small number of primes.

Mixing existing ideas is enough to achieve this.

- Borrow from the structure of number field root finding.
- Use subproduct trees very often.

Setup

Let $\{p_i\}$ be a collection of primes totally split in K.

Facts with CRT modulo prime ideals

We let $(p_i) = \mathfrak{p}_{i,1} \cdots \mathfrak{p}_{i,d}$, and $P = \prod p_i$.

- CRT: Knowing $\{T \bmod \mathfrak{p}_{i,j}\}$, we can recover $T \bmod P$.
- Each of the coefficients of A above is expressed linearly.

Indeed we may write polynomials $Q_{i,j}(x)$ such that:

$$T(x) \equiv \sum_{i,j} Q_{i,j}(x) T_{i,j} \mod P.$$

It is also posssible to do the same with the p_i-adic lifts of $T_{i,j}$.

CRT and lifting together

Let p be totally split in K. We have:

$$\mathbb{Z}[\alpha] \hookrightarrow K_p \cong \mathbb{Q}_p \times \cdots \times \mathbb{Q}_p.$$

For computing $\pm\sqrt{S(\alpha)}$ in all the $\mathbb{Z}_p$ parts, one may:

- Compute the adic roots of f modulo p: $r_1, \ldots, r_d$.
- Lift each p-adically to some precision: $\tilde{r}_1, \ldots, \tilde{r}_d$.
- Deduce the image of $S(\alpha)$ in each part.
- Compute the p-adic square root (by lifting).

If we do so modulo many primes p_i, we can recover the result as in the CRT setup.
Lift up to precision λ modulo each p_i, then:

$$T(x) \equiv \sum_{i,j} \pm Q_{i,j}(x)\, T_{i,j} \mod P^{\lambda}.$$

Finding the correct combination

Issue already encountered:

many CRT shares $\Rightarrow$ intractable recombination problem

- This is the reason why we focus on relatively few primes (number of shares $k = d \times \#\{p_i\} \lessapprox 60$).
- Improved reconstruction allows more parallelism.

Streamlining reconstruction

Let M be a bound on the coefficients of $T(x)$, and let $P^\lambda > M^{\frac{1}{\epsilon}}$.

$$T(x) \equiv \sum_{i,j} \pm Q_{i,j}(x)\, T_{i,j} \mod P^\lambda,$$

$$\underbrace{\frac{1}{P^\lambda} T(x)}_{\text{coeffs} < \epsilon} \equiv \pm \frac{Q_{i,j}(x)\, T_{i,j}}{P^\lambda} \mod 1.$$

Problem reduced to (e.g.) finding a sum of floating-point values close to an integer.

- Trivially implementable in $O(2^{k/2})$ for k shares.
- Can go up to $k \approx 60$ easily.
- Best complexity $O(2^{0.313k})$.

Overcoming obstacles

For computing $\{S(\alpha) \bmod \mathfrak{p}_{i,j}\}$, use subproduct trees.

- Compute $S(\alpha)$.
- Compute P^{λ} as a subproduct tree.
- Make $S(\alpha)$ descend along the tree.

We achieve the same complexity as the lifting approach.

- We have waived the inert prime assumption.
- Possible to parallelize: on t^2 nodes, t-fold reduction in both time and space complexity on each node.

Results

Algorithm implemented in CADO-NFS.

- Has been used to (re-)do the sqrt for RSA-768.
  - Computation time 6 hours on 144 Intel cores.
  - Montgomery: 4 hours on 144 AMD cores.
- Has also been used for a 190-digit SNFS.

Conclusion

Asymptotically fast algorithms are fast.

- Lifting approach has some limitations:
  - Needs inert primes.
  - Memory-hungry (probably soon no longer a problem).
- New CRT approach.
  - Nice parallelizable version of the lifting approach.
  - Waives the number field assumptions.
  - Competitive with Montgomery's algorithm.

On the other hand, code size in the end almost invalidates the "lazy programmer" assumption that this is more pragmatic than coding all the background for Montgomery's algorithm.