Proving the primality of very large numbers with fastECPP

ANTS VI, University of Vermont in Burlington

Wednesday 16 June, 2004

J. Franke, T. Kleinjung, F. Morain, T. Wirth

{**franke,thor,wirth**}**@math.uni-bonn.de**

morain@lix.polytechnique.fr

Plan

I. Introduction.

II. (fast)ECPP: theory and analyses.

III. Old and new features[a].

IV. Distributed implementations.

V. Some proofs.

VI. Conclusions.

[a] includes the now famous FKMW algorithm for detecting smooth numbers as advertised by Dan Bernstein in his even more famous talk of Monday 14

I. Introduction

Goal: decide whether N is prime or composite.

CS/Crypto answer: Solovay-Strassen; Artjuhov-Miller-Rabin.

Using Watkins's classification:

- Algorithmic primality proving: Gauss sums; Goldwasser-Kilian; Adleman-Huang; Agrawal-Kayal-Saxena.
- Computational primality proving: Jacobi sums; ECPP;

ECPP

Brief history: anticipated by Bosma, Chudnosky & Chudnovsky (1985); invented by Atkin around 1986; Atkin+FM; etc.

Features: ECPP is of LVMH type

(i) It answers prime, composite, or I don't know. Always explains why (proofs of polynomial length are given).

(ii) It runs in heuristic polynomial time $O((\log N)^{5+\mu})$, and gives a certificate of length $O((\log N)^2)$ that can be checked in time $O((\log N)^{3+\mu})$.

Rem. $O((\log N)^{1+\mu}) =$ time needed to multiply $\log N$-bit integers, $0 \leq \mu \leq 1$.

The Fermat legacy

Thm. N is prime iff $(\mathbb{Z}/N\mathbb{Z})^*$ is cyclic of order $N-1$:

$$a^{N-1} \equiv 1 \bmod N$$

$$\forall \text{ prime } p \mid N-1,\ a^{\frac{N-1}{p}} \not\equiv 1 \bmod N$$

Certificate: $(N, \{p \mid N-1\}, a)$.

Thm. (**Pocklington**, 1914) Let $s \mid N-1$ s.t. $s > \sqrt{N}$ and

$$a^{N-1} \equiv 1 \bmod N$$

$$\forall \text{ prime } q \mid s, \gcd(a^{\frac{N-1}{q}} - 1, N) = 1$$

Then N is prime.

II. (fast)ECPP: theory and analyses

$\mathbf{K} = \mathbb{Q}(\sqrt{-D})$, $\mathcal{O}_K = \mathbb{Z}[\omega]$ ring of integers of $\mathbf{K}$.

Thm. (Siegel) $h(-D) = \#Cl_K = O(D^{1/2+\varepsilon})$.

Def. Hilbert class field: $\mathbf{K}_H$ is the maximal unramified abelian extension of $\mathbf{K} = \mathbb{Q}(\sqrt{-D})$.

Thm. $\mathbf{K}_H = \mathbf{K}(j(\omega))$, where $j(q) = 1/q + 744 + \cdots$;
$\Sigma_H = \mathrm{Gal}(\mathbf{K}_H/\mathbf{K}) \simeq Cl_K$ (action is completely explicit).

Fundamental Thm. $4p = U^2 + DV^2$ iff (p) splits completely in $\mathbf{K}_H$. The probability for this to happen is $1/(2h)$.

Elliptic curves with complex multiplication

Let $\overline{E}/\mathbb{Q}(j(\omega))$ be a curve of invariant $j(\omega)$, (a so-called curve with CM by $\mathcal{O}_K$).

If $4p = U^2 + DV^2$, then we can reduce $\overline{E}$ modulo one of the factors (of degree 1) of (p) in $\mathbf{K}_H$ to get $E/\mathbb{F}_p$ s.t.

$$\#E(\mathbb{F}_p) = p + 1 - U$$

(and conversely).

From a practical point of view: $H_D(X)$ has h roots modulo p; any one gives a suitable E, and this is easy to build.

Computing $H_D(X)$: the global picture

Replace j by other invariants to get small polynomials (Weber, Schertz/Enge, Enge/FM – ANTS V, Gee/Stevenhagen, etc.).

Methods:

- good old floating point complex numbers (..., AOLAtkin+FM, AEnge+FM, ...): $\tilde{O}(h^2)$ (Enge04; see also R. Dupont for the fast evaluation of η).
- rather new p-adic methods (Couveignes/Henocq – ANTS V, Bröker/Stevenhagen, Lercier/Riboulet-Deyris): $\tilde{O}(h^2)$.

ECPP $=$ Pocklington $+$ CM curves

[Step 1.] find $-D$ s.t. $4N = U^2 + DV^2$ and $m = N + 1 \mp U = 2N'$ with N' prp.

[Step 2.] Build $\overline{E}/\overline{\mathbb{Q}}$ having CM by $\mathcal{O}_K$ (independent of N).

[Step 3.] Reduce $\overline{E}$ modulo N to get E.

[Step 4.] Find $P \neq O_E$ on E s.t. $[N']P = O_E$. If a factor f of N is found, return ("composite", f). If such a point cannot be found in reasonable time, return ("I don't know", "no P found").

[Step 5.] N is "prime"; $N \leftarrow N'$ and goto Step 1.

Complexity: the easy pieces

$L = \log N$; multiplication is $L^{1+\mu}$, $0 \leq \mu \leq 1$.

[Step 2.] Build $\overline{E}/\overline{\mathbb{Q}}$ having CM by $\mathcal{O}_K$: $\tilde{O}(h^2)$.

[Step 3.] Reduce $\overline{E}$ modulo N to get E: (Cantor-Zassenhaus) $O((\log h)LM(h)L^{1+\mu})$.

[Step 4.] Find $P \neq O_E$ on E s.t. $[N']P = O_E$: $O(L^{2+\mu})$.

[Step 5.] N is "prime"; $N \leftarrow N'$ and goto Step 1. $\Rightarrow O(L)$ steps.

Zooming on Step 1

for $D \in \mathcal{D}$

if $\exists\, U, V \in \mathbb{Z}$ s.t.

$$4N = U^2 + DV^2 \tag{1}$$

then if $m = N + 1 \mp U = 2N'$ with N' prp **then** goto Step 2.

return ("I don't know", "not enough D's?").

Costs: Cornacchia needs $\sqrt{-D} = (O(L^{2+\mu}))$ + HEEA ($O(L^{1+\mu})$); prp $= O(L^{2+\mu})$.

Probability of (1) solvable is $1/(2h)$.

Conj. Probability of m of the good form is $1/\log N = 1/L$.

Hypothesis: let a be an integer and $\mathcal{D} = \{D \leq O(L^a)\}$. (Note that $\#\mathcal{D} = O(L^a)$.)

We expect to use $O(L^{a/2+1})$ discriminants for a total cost

$$\underbrace{O(L^{a/2+1}\ (L^{2+\mu} + L^{1+\mu}))}_{\text{Cornacchia}} + \underbrace{O(L^1\ L^{2+\mu})}_{\text{prp}}$$

$$+\underbrace{O((\log h)M(h)L^{2+\mu})}_{\text{CZ}}.$$

$a = 2$: $D = O(L^2)$, $h = O(L)$.

Cost of all $\sqrt{-D}$: $O(L^2 \cdot L^{2+\mu})$ – this is the dominant term!

Conj. ECPP costs $O(L \cdot L^2 \cdot L^{2+\mu})$, or $O((\log N)^{5+\mu})$.

fastECPP

Genesis:

• A. K. Lenstra & H. W. Lenstra, Jr.: Algorithms in number theory in HANDBOOK OF THEORETICAL COMPUTER SCIENCE, 1990: sketch analysis of ECPP and cite Shallit's improvement.

• During FM's algorithmic trip for Bourbaki: dig the paper (fall 2002).

• Exchange of emails with Dan Bernstein following his improvement on AKS, discussing ECPP alleged complexity (spring 2003).

• Why not try in FM's new mpi ECPP implementation? (spring 2003 ff).

Key idea: decrease the cost of $\sqrt{-D} \bmod N$ by combining them.

Genus theory: let $0 > -D = \prod_{i=1}^{t} q_i^*$ be a fundamental discriminant, odd part squarefree ($D \equiv 3 \bmod 4$ or $D \equiv 4, 8 \bmod 16$): where $q_i^* \in \{-4, \pm 8, = (-1/q_i)q_i$ for q_i an odd prime$\}$.

Ex. $-20 = (-4) \times 5$, $-40 = (-8) \times 5$, $-24 = (-3) \times 8$, $-15 = (-3) \times 5$.

Prop. $4p = U^2 + DV^2$ in rational integers U and V only if $\left(\frac{q_i^*}{p}\right) = +1$ for all i.

The algorithm

[Step 1'.] a. Find the r smallest primes q^* s.t. $\left(\frac{q^*}{N}\right) = 1$, forming $\mathcal{Q} = \{q_1^*, q_2^*, \ldots, q_r^*\}$ and compute all $\sqrt{q_i^*} \bmod N$.

b. Use $\mathcal{D} = \{-D = (-q_{i_1})q_{i_2}\}$.

Conj. The complexity of fastECPP is $O((\log N)^{4+\mu})$.

"Proof": take $r = O(L) \Rightarrow \#\mathcal{D} = O(L^2)$.

Computing the r squareroots: $O(L \cdot L^{2+\mu})$.

All Cornacchia: $O(L^2 \cdot L^{1+\mu})$. All prp tests: $O(L \cdot L^{2+\mu})$. $\square$

Rem. Trivial generalization (and practical improvement): use $O(2^r)$ subsets $\{q_{i_1}^*, q_{i_2}^*, \ldots, q_{i_s}^*\}$.

III. Old and New features

Old and new tactics

Fast Fourier Transform in the West: twice as fast as GMP.

Atkin's trick: $\mathrm{pgcd}(X^{(N-1)/k} - \zeta_k^i, H_D(X))$. Very often, only one exponential.

TCMM: (Mihăilescu's thesis) optimisation of $A(X)B(X) \bmod (N, M(X))$. Brutal speedup of 2. Very well suited for $\log N \gg \deg(M)$.

Smooth class numbers

Speedup Step 3. Use Galois action to precompute the factorisation of $H_D(X)$ over a tower of extensions (see Hanrot + M. ISSAC 2001; Enge+M. AAECC 2003).

Bröker's example: $D = 79,580,203$; $h = 1536 = 2^9 \cdot 3$; using $w_{7,13}$: precision is 8269 bits. Time for building $H_D[w_{7,13}](X)$: 43 sec on a 2.6 GHz Xeon (GMP+mpfr+mpc); Galois decomposition of fields (nine degree 2 equations, one degree 3): 31 sec.

Largest examples: $(D; h; T) \in (806,492,231; 60284; 4d)$, $\{(4,744,754,740 > 2^{32}; 15360 = 2^{10} \cdot 3 \cdot 5; 2.6d)$, $(294,699,719; 24444 = 2^2 \cdot 3^2 \cdot 7 \cdot 97; 1d)\}$.

Finding the new N'

Of course, we look for $m = fR$, with f B-smooth and R having prime factors $> B$, hoping that R is a probable prime.

Let's forget that m is a norm in $\mathbf{K}$.

Prop. Proba(R prime) $\rightarrow \exp(\gamma)\log B/\log m$.

Hence, number of m's to be tried is
$t \approx \exp(-\gamma)\log N/\log B = O(L/\log B)$.

Prop. $N/N' \approx \log B$.

Early abort strategy

Idea: force $N/N' > 2^\beta$.

Number	$10^{2999}+1887$		$10^{3999}+4771$		$10^{4999}+22669$	
Step 1	81 h	58 h	280 h	164 h	716 h	476 h
Step 2-4	26 h	26 h	76 h	86 h	209 h	261 h
H_D	1680 s	4880 s	4497 s	7317 s	3 h	5 h
CZ	22 h	22 h	63 h	75 h	179 h	234 h
# steps	436	358	597	437	734	512
$\max h$	1968	2336	2184	2432	3400	4000
$\overline{h}$	86	116	120	164	152	272

Finding small divisors of a lot of m_i's

Pb: let $m_1, \ldots, m_t$ be integers of size L. Find an algorithm that strips the primes up to B from the numbers m_i's.

Traditionally: (or for small numbers)

1. precompute $r_p = (N+1) \bmod p$ for $p \leq B$. $O(\pi(B)L^{1+\varepsilon})$

2. for $i = 1..t$, for all p, compute $u_p = |U| \bmod p$ and compare with r_p. $O(\pi(B)(t/2/2)L^{1+\varepsilon})$.

Total cost for N': $O(\pi(B)tL^{1+\varepsilon}) + O(tL^{2+\mu})$.

If $t = O(L/\log B)$ and we balance the two quantities, then $B = O(L^1)$ for a cost of $O(L^{3+\mu})$, and a gain of $\log\log N$ bits.

The FKMW algorithm

Looks like Bernstein00, but is not a variant of it...

1. Build the product $P = \prod_{p \leq B, p \text{ prime}} p$ using a binary tree. $O(B(\log B)^2)$
2. Compute $\tilde{m}_i = P \bmod m_i$ using a remainder tree (i.e., $P \bmod (m_1 m_2) \rightarrow P \bmod m_1, P \bmod m_2$).
3. [Bernstein04] compute $y_i = (\tilde{m}_i)^{2^e} \bmod m_i$ with $2^{2^e} \geq m_i$; $\gcd(y_i, m_i)$ is the desired quantity.

Cost: $O(B(\log B)^{2+\varepsilon}) + O(tL^{2+\mu}) \Rightarrow B = O(L^3)$.

The FKMW algorithm

Looks like Bernstein00, but is not a variant of it...

1. Build the product $P = \prod_{p \leq B, p \text{ prime}} p$ using a binary tree. $O(B(\log B)^2)$
2. Compute $\tilde{m}_i = P \bmod m_i$ using a remainder tree (i.e., $P \bmod (m_1 m_2) \rightarrow P \bmod m_1, P \bmod m_2$).
3. [Bernstein04] compute $y_i = (\tilde{m}_i)^{2^e} \bmod m_i$ with $2^{2^e} \geq m_i$; $\gcd(y_i, m_i)$ is the desired quantity.

Cost: $O(B(\log B)^{2+\varepsilon}) + O(tL^{2+\mu}) \Rightarrow B = O(L^3)$.

Rem. can be distributed in a variety of ways ($B = 2^{32}$).

IV. Distributed implementations

Strategy 1

S1. The master sends to each slave a batch of $\sqrt{q_i^*}$ to compute.

S2. Each slave computes its batch and sends the results back to the master.

S3. The master sends all the squareroots to all the slaves, so that each slave can compute any $\sqrt{-D}$ that is needed.

S4. The master sends batches of D's to the slaves. Each slave is responsible for the resolution of (1) and the factorization of the m's. If one is found, it is sent back to the master which checks the results and starts a new phase.

Load balancing

By genus theory, each D with t prime factors has splitting probability $g(-D)/h(-D)$ with $g(-D) = 2^{t-1}$.

$\rightarrow$ sort D's in increasing value of h/g.

$\rightarrow$ $\sum_j g(D_{i_j})/h(D_{i_j})$ is the expected number of splitting D's, hence make it the same for all the slaves.

Strategy 2 (more intricate on the master's side)

More intricate on the master's side, easy for the slave:

T1. Check whether the class number for a discriminant D is good.

T2. Compute $\sqrt{q_i^*} \bmod N$.

T3. Try to solve (1) for a given D.

T4. Do trial division for an interval of primes and a batch of m's and return the factored parts.

T5. Do a prp test.

The master keeps lists of tasks of these five types, and creates new tasks depending on priorities, T1 lowest and T5 highest.

V. Some proofs

$(x, y; \#dd)$	(2177, 580; 6016)	(2589, 218; 6055)	(2551, 622; 7127)
# steps	801	736	965
Step 1 (CPU)	164 d	103 d	235 d
Step 1 (WCT)	164×1.2	103×1.1	235×1.3
$\sqrt{-q^*}$	81 d	30 d	74 d
Steps 2-4	28 d	21 d	55 d
H_D	2951 sec	1686 sec	18451 sec
CZ	26 d	20 d	50 d
Check	25 h	22 h	45 h
$\max h$	1980	2080	3312
$\bar{h}$	121	103	190
$\max D$	7,749,263	19,076,479	52,396,648

From P. Leyland's tables of $x^y + y^x$; 6 biproc Xeon at 2.66 GHz

Reproving $(2177, 580)$ (6016dd)

date	030513-030604	040524-040528
# steps	801	588
Step 1 (CPU)	164 d	25 d
$\sqrt{-q^*}$	81 d	6 d
classno	0	3 d
Sieve / $P \bmod M$	17 d	2 d
PRP	41 d	15 d
Steps 2-4	28 d	10 d
H_D	2951 sec	1 d
CZ	26 d	8 d
$\max h$	1980	8448
$\bar{h}$	121	460
$\max D$	7,749,263	876,695,224

Franke, Kleinjung, Wirth found a new frontier!

$$N = 10^{9999} + 33603$$

030717–030819: implementation using pvm; used six 900MHz PIII CPUs, then moved to 12 nodes of parnass2 (LINUX cluster).

Step 1: 234.5 days on a 1GHz Pentium; $B = 2^{32}$.

Step 2-4: 140 days on a single Athlon 800MHz.

$(x, y, \#dd)$	(2438, 1995; 8046)	(3571, 648; 10041)	(2680, 2319; 9020)
date	0715-0901	1001-1220	0320–0419
# steps	1128	947	826
Step 1 (CPU)	355 d	531 d	179 d
Step 1 (WCT)	355×1.13	531×1.2	
$\sqrt{-q^*}$	138 d	204 d	59 d
classno			11 d
PRP			102 d
$P \bmod M$			8 d
Steps 2-4	77 d	138 d	32 d
H_D/CZ	6h/69 d	6h/124 d	38 h/27 d
Check	70 h	85 h	28 h
$\max h/\overline{h}$	3640/209	6176/409	11616/645
$\max D$	87,949,348	95,895,480	958,024,452

VI. Conclusions

In brief: 1000dd (20 min), 2000dd (6 h = 360 min), 6000dd (1 month).

Did we cheat somewhere? Maybe we increased the probability of the program answering ”I don’t know”.

What next? all phases have the same asymptotical cost, seems hard to improve all of them! Optimize again and again to decrease the constants (fastCornacchia, etc.).

More and more difficult to optimize: too many parameters.

What about competition? same complexity as Bernstein/Mihăilescu probabilistic version of AKS. Remaining advantage for ECPP: time for checking the certificate is $\tilde{O}((\log N)^3)$.

Lessons from supporting ECPP these last 15 years

- 1992: 1505dd; 2003: 10,000dd; 2004: ??.
- $\Rightarrow$ hegemony is bad: stimulation from the Wind* world and its crazy prime provers (cf. PRIMO of Marcel Martin).
- One lesson: more and more powerful computers + GMP $\Rightarrow$ fast methods begin to be fast in the real life (really fast available multiplication of integers, fast algorithms from computer algebra, etc.) $\Rightarrow$ larger and larger (D, h) the key to fastECPP.
- Strategies for large numbers can be against intuition (remember Dan’s talk). Shall we design a mesh ECPP ???