20 years of ECM

Paul Zimmermann

LORIA/INRIA Lorraine, 615 rue du jardin botanique, BP 101, F-54602 Villers-lès-Nancy, France,
zimmerma@loria.fr

Abstract. The Elliptic Curve Method for integer factorization (ECM) was invented by H. W. Lenstra, Jr., in 1985 [13]. In the past 20 years, many improvements of ECM were proposed on the mathematical, algorithmic, and implementation sides. This paper summarizes the current state-of-the-art, as implemented in the `GMP-ECM` software.

Introduction

Before ECM was invented by H. W. Lenstra, Jr., in 1985 [13], Pollard's ρ algorithm and some variants were used, for example to factor the eighth Fermat number F_8 [8]. As soon as ECM was discovered, many researchers worked hard to improve the original algorithm or efficiently implement it. Most current improvements to ECM were already invented by Brent and Montgomery in the end of 1985 [5, 17][1].

In [5], Brent describes the "second phase" in two flavours, the "P-1 two-phase" and the "birthday paradox two-phase". He already mentions Brent-Suyama's extension, and the possible use of fast polynomial evaluation in stage 2, but does not yet see how to use the FFT. At that time (1986), ECM could find factors of about 20 digits only; however Brent predicted: "*we can forsee that p around* 10^{50} *may be accessible in a few years time*". This happened in September 1998, when Conrad Curry found a 53-digit factor of $2^{677} - 1$ with Woltman's MPRIME program. According to Fig. 1, which displays the evolution of the ECM record since 1991, and extrapolates it using Brent's formula $\sqrt{D} = (Y - 1932.3)/9.3$, a 100-digit factor — which corresponds to the current GNFS record (RSA-200) — could be find by ECM around 2025, i.e., in another 20 years.

In [17], Montgomery gives a unified description of P-1, P+1 and ECM. He already mentions the "FFT continuation" suggested by Pollard for P-1. A major improvement was proposed by Montgomery with the "FFT extension" [18], which enables one to speed up significantly stage 2.

Several efficient implementations have been made, in particular by Brent [6], Montgomery (ECMFFT), and Woltman (PRIME95/MPRIME). This allowed Brent to find 21-digit and 22-digit factors of the eleventh Fermat number F_{11} in 1988.

Many large factors have been found by ECM. Among others we can cite the factorization of the tenth Fermat number [7]:

$$F_{10} = 45592577 \cdot 6487031809 \cdot 4659775785220018543264560743076778192897 \cdot p_{252}.$$

The smallest unfactored Fermat number, F_{12}, being out of reach for NFS-based methods (Number Field Sieve), the main hope to factor it relies on ECM.

The aim of this paper is to describe the state-of-the-art in the ECM domain, and in particular the algorithms implemented in the GMP-ECM software. §1 recalls the ECM algorithm and defines

[1] The first version of Brent's paper is from September 24, 1985 — revised December 10, 1985 — and Montgomery's paper was received on December 16, 1985.

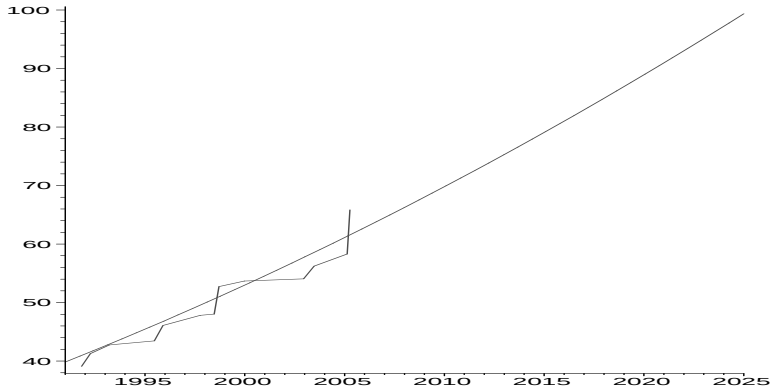

Fig. 1. Graph of ecm records since 1991 (digits vs year), and extrapolation until 2025.

the notations used in the rest of the paper, while §2 describes the algorithms used in Stage 1 of ECM, and §3 those in Stage 2. Finally, §4 exhibits nice factors found by ECM, and discusses further possible improvements.

1 The ECM method

Notations. In the whole paper, n denotes the number to be factored, p a (possibly unknown) prime factor of n, and π a prime; the function $\pi(x)$ denotes the number of primes less than or equal to x. All arithmetic operations are implicitely performed modulo n. We assume n has l words in the machine word base β — usually $\beta = 2^{32}$ or 2^{64} —, i.e., $\beta^{l-1} \leq n < \beta^l$. Depending on the context, we write $M(d)$ for the cost of multiplying two d-bit integers, or two degree-d polynomials — where operations on the coefficients count $O(1)$.

This section is largely inspired from [7] and [17]. Consider a field K of characteristic other than 2 or 3. The elliptic curve $E_{A,B}$ is the set of points $(X, Y) \in K$ such that

$$Y^2 = X^3 + AX + B,$$

where $A, B \in K$, $4A^3 + 27B^2 \neq 0$, plus a special "point at infinity" denoted O_E. $E_{A,B}$ admits a group structure, where the addition of two points can be effectively computed.

For a computer implementation, it is more efficient to use Montgomery's form,

$$by^2 = x^3 + ax^2 + x,$$

which can obtained from Weierstrass form above by the change of variables $X \to (3x+a)/(3b), Y \to y/b, A \to (3 - a^2)/(3b^2), B \to (2/9a^3 - a)/(3b^3)$. Moreover, one usually prefers an homogeneous form:

$$by^2z = x^3 + ax^2z + xz^2, \tag{1}$$

where the triple $(x : y : z)$ represents the point $(x/z : y/z)$ in affine coordinates.

The ECM method starts by choosing a random curve $E_{a,b}$ and a random point $(x : y : z)$ on it. All computations are done modulo the number n to factor, as if $\mathbb{Z}/n\mathbb{Z}$ was a field. The only

operation which may fail is when computing the inverse of a residue x modulo n, if $\gcd(x, n) \neq 0$. But then a factor of n was found, the program outputs it and exits.

Here is a high-level description of the ECM algorithm:

Algorithm ECM.
Input: a number n, integer bounds $B_1 \leq B_2$.
Output: a factor of n, or FAIL.
Choose a random elliptic curve $E_{a,b} \bmod n$ and a point $P_0 = (x_0 : y_0 : z_0)$ on it.
[Stage 1] Compute $Q := \prod_{\pi \leq B_1} \pi^{\lfloor (\log B_1)/(\log \pi) \rfloor} P_0$ on $E_{a,b}$
[Stage 2] for each prime π, $B_1 < \pi \leq B_2$,
compute $(x_\pi : y_\pi : z_\pi) = \pi Q$ on $E_{a,b}$
$g \leftarrow \gcd(n, z_\pi)$
if $g \neq 1$, output g and exit
output FAIL.

Suyama's parametrization. Suyama's parametrization is widely used, and therefore enables one to reproduce factorizations found by different programs. Choose a random integer $\sigma > 5$; usually a random 32-bit value is enough, but when running many curves on the same number, one might want to use a larger range. Then compute $u = \sigma^2 - 5$, $v = 4\sigma$, $x_0 = u^3 \bmod n$, $z_0 = v^3 \bmod n$, $a = (v-u)^3(3u+v)/(4u^3v) - 2 \bmod n$. One can check that Eq. (1) holds with for example $g = u/z_0$ and $y_0 = (\sigma^2-1)(\sigma^2-25)(\sigma^4-25)$. In fact, the values of g and y are not needed; all the arithmetic operations involve x and z only. Indeed, for a given pair (x, z), only two values of y give a valid point $(x : y : z)$ on $E_{a,b}$ according to Eq. (1). Since those values are y and $-y$, ignoring the y-coordinate identifies P and $-P$. As will be seen later, this is precisely what we want.

1.1 Why does ECM work?

Let p be a prime factor of n, and consider the elliptic curve $E_{a,b} \bmod p$. Hasse's theorem says that the order g of $E_{a,b} \bmod p$ satisfies

$$|g - (p+1)| < 2\sqrt{p}.$$

When a and b vary, g essentially behaves as a random integer in $[p+1-2\sqrt{p}, p+1+2\sqrt{p}]$, with some additional conditions imposed by the type of curve chosen (for example Suyama's parametrization ensures g is divisible by 12).

ECM will find the factor p — which is not necessarily the smallest factor of n — when g is (B_1, B_2)-smooth, i.e., when the largest prime factor of g is less or equal to B_2, and its second smallest prime factor less or equal to[2] B_1. The factor p will be found in stage 1 when g is (B_1, B_1)-smooth, and in stage 2 otherwise.

Remark. If two or more factors of n have a (B_1, B_2)-smooth group order for the chosen curve, they will be found simultaneously, which means that ECM will output their product, which can even be n if all its prime factors have a (B_1, B_2)-smooth group order. This should not be considered as a failure: instead restart the same curve with smaller B_1, B_2 to split the different prime factors.

[2] The definition of (B_1, B_2)-smoothness used in Algorithm ECM above and by most software is slightly different: all primes $\pi \leq B_1$ should appear to a power $q^k \leq B_1$, and similarly for B_2; in practice this makes little difference.

1.2 Complexity of ECM

The expected time used by ECM to find a factor p of a number n is

$$O(L(p)^{\sqrt{2}+o(1)}M(\log n)),$$

where $L(p) = e^{\sqrt{\log p \log\log p}}$, and $M(\log n)$ representes the complexity of multiplication modulo n. The second stage enables one to save a factor of $\log p$ — which is absorbed by the $o(1)$ term above. Mathematical and algorithmic improvements act on the $L(p)^{\sqrt{2}+o(1)}$ factor, while arithmetic improvements act on the $M(\log n)$ factor.

2 Stage One

Stage 1 computes $Q := \prod_{\pi \le B_1} \pi^{\lfloor (\log B_1)/(\log \pi)\rfloor} P_0$ on $E_{a,b}$. That big product is not computed as such. Instead, we use the following loop:

$Q \leftarrow P_0$
for each prime $\pi \le B_1$
    compute k such that $\pi^k \le B_1 < \pi^{k+1}$
    for $i := 1$ to k do
        $Q \leftarrow \pi \cdot Q$.

The multiplication $\pi \cdot Q$ on the elliptic curve is done using additions ($P + Q \to P + Q$) and duplications ($P \to 2P$).

To add two points $(x_P : : z_P)$ and $(x_Q : : z_Q)$, one uses the following formula, where $(x_{P-Q} : : z_{P-Q})$ corresponds to the difference $P - Q$:

$$x_{P+Q} = 4z_{P-Q} \cdot (x_P x_Q - z_P z_Q)^2, \qquad z_{P+Q} = 4x_{P-Q} \cdot (x_P z_Q - z_P x_Q)^2.$$

This can be computed using 6 multiplications (among which 2 squares) as follows:

$$\begin{array}{ll} u \leftarrow (x_P + z_P)(x_Q - z_Q) & v \leftarrow (x_P - z_P)(x_Q + z_Q) \\ w \leftarrow (u + v)^2 & t \leftarrow (u - v)^2 \\ x_{P+Q} \leftarrow z_{P-Q} \cdot w & z_{P+Q} \leftarrow x_{P-Q} \cdot t. \end{array}$$

To duplicate a point $(x_P : : z_P)$, one uses the following formula:

$$x_{2P} = (x_P^2 - z_P^2)^2, \qquad z_{2P} = (4x_P z_P)[(x_P - z_P)^2 + b(4x_P z_P)], \tag{2}$$

where $b = (a+2)/4$, with a from Eq. (1). This formula can be implemented using 5 multiplications (including 2 squares) as follows:

$$\begin{array}{lll} u \leftarrow (x_P + z_P)^2 & v \leftarrow (x_P - z_P)^2 & t \leftarrow b(u - v) + v \\ x_{2P} \leftarrow uv & z_{2P} \leftarrow (u - v)t. & \end{array}$$

Since the difference $P-Q$ is needed to compute $P+Q$, this is a special case of additions chains, called "Lucas chains" by Montgomery, who designed an heuristic algorithm "PRAC" to compute them [15] (see §2.2).

2.1 Residue Arithmetic

To obtain an efficient implementation of ECM, an efficient underlying arithmetic is important. The main operations to be performed are additions, subtractions and multiplications modulo the number n to be factored. Other operations (divisions, gcds) are seldom, or can be replaced by modular multiplications. Since additions and subtractions have cost $O(\log n)$, the main operation to be optimized is the modular multiplication: given $0 \leq a, b < n$, compute $c = ab \bmod n$.

We distinguish two cases: classical $O(\log^2 n)$ arithmetic, and subquadratic arithmetic. On a Athlon XP 1700+, GMP-4.1.4 switches to Karatsuba's algorithm up from 26 words, i.e., about 240 decimal digits. Since ECM is often used to factor numbers smaller than this, it is worth optimizing classical arithmetic.

For special numbers, like factors of $\beta^k \pm 1$, one may use ad-hoc routines. Assume for example $dn = \beta^k - 1$. The product $c = ab$ of two residues can be reduced as follows: write $c = c_0 + c_1\beta^k$, where $0 \leq c_0, c_1 < \beta^k$; then $c = c_0 + c_1 \bmod n$. Instead of reducing c, a $2l$-words integer (recall n has l words), we reduce $c_0 + c_1$, which has k words only (plus possibly one carry bit). Alternatively, if the cofactor d is small, one can reduce c modulo $\beta^k - 1$ only, and perform multiplications on k words instead of l words. (Indeed, a canonical representation is not needed.) GMP-ECM implements such a special reduction for large divisors of $2^k \pm 1$, using the latter method. It also uses special code for Fermat numbers $2^{2^k} + 1$: indeed, GMP fast multiplication code precisely uses Schönhage-Strassen algorithm, i.e., multiplication modulo $2^m + 1$ [20].

Efficient Assembly Code. While using clever high-level algorithms may give a speedup of 10% or 20%, at the expense of several months to invent and implement those algorithms, a twofold speedup may be obtained in one day, just rewriting one of the assembly routines for integer arithmetic[3].

GMP-ECM is based on the GNU MP library (GMP for short) [10], thus benefits from the portability of GMP, and from the efficiency of its assembly routines (the `mpn` layer). A library dedicated to modular arithmetic — or even better to computations on elliptic curves — might yet be faster. Since all operations are done on numbers of the same size, we might use a library with special assembly code for each word size, up to the Karatsuba threshold.

Quadratic Arithmetic. In the quadratic domain, up to 200−300 digits according to the processor, the best solution is to use Montgomery representation [16]: The number n to be factored having l words in base β, each residue a is replaced by $a' = \beta^l a \bmod n$. Additions and subtractions are unchanged, and multiplications are replaced by the REDC operation: $\mathrm{REDC}(a, b) := ab\beta^{-l} \bmod n$. This operation can be efficiently implemented on modern computers, and does not require any correction unlike the classical division.

There are two ways to implement REDC: (i) either interleave the multiplication and the reduction as in algorithm MODMULN from [17], (ii) or perform them separately. GMP-ECM uses the latter way, which enables it to use the efficient GMP assembly code for base-case multiplication. One first computes $c = ab$, having at most $2l$ words in base β. The reduction $r := c \bmod n$ is performed with the following GMP code (which is exactly that of version 6.0.1 of GMP-ECM, with variable names changed to match the above notations):

[3] The author indeed noticed a speedup of more than 2 with GMP-ECM, when Torbjörn Granlund rewrote the UltraSparc assembly code for GMP.

```
static void
ecm_redc_basecase (mpz_ptr r, mpz_ptr c, mpmod_t modulus)
{
  mp_ptr rp = PTR(r), cp = PTR(c);
  mp_srcptr np = PTR(modulus->orig_modulus);
  mp_limb_t cy;
  mp_size_t j, l = modulus->bits / __GMP_BITS_PER_MP_LIMB;

  for (j = ABSIZ(c); j < 2 * l; j++)
    cp[j] = 0;
  for (j = 0; j < l; j++, cp++)
    cp[0] = mpn_addmul_1 (cp, np, l, cp[0] * modulus->Nprim);
  cy = mpn_add_n (rp, cp, cp - l, l);
  if (cy != 0)
    mpn_sub_n (rp, rp, np, l);
  MPN_NORMALIZE (rp, l);
  SIZ(r) = SIZ(c) < 0 ? -l : l;
}
```

The main idea — independently discovered by Kevin Ryde, another GMP developer — is to store the carry words from `mpn_addmul_1` in the low l words of c, just after they are set to zero by REDC. In such a way, one replaces l expensive carry propagations by one call to `mpn_add_n`.

Subquadratic Arithmetic. For large numbers, a subquadratic arithmetic is needed. Again, one can use either the classical representation, or Montgomery representation. In both cases, the best known algorithms require $2.5M(l)$ for a l-word modular multiplication: $M(l)$ for the multiplication $c := ab$, and $1.5M(n)$ for the reduction $c \bmod n$ using Barrett's algorithm [1], or its least-significant-bit variant for $c\beta^{-l} \bmod n$. LSB-Barrett is exactly REDC, with a big word β^l [19]: after the precomputation of $m = -n^{-1} \bmod \beta^l$, compute $d = cm \bmod \beta^l$, and $(c + dn)\beta^{-l}$. Since all reductions are done modulo the same n, the precomputation of m is amortized and does not count in the average cost. The $1.5M(n)$ reduction cost is obtained using the "wrap-around" trick for the last multiply dn (see §3.2), since the low part is known to be equal to $-c \bmod \beta^l$.

2.2 Evaluation of Lucas Chains

A Lucas chain is an addition chain in which the sum $i + j$ of two terms can appear only if $|i - j|$ also appears. (This condition is needed for the point addition in homogeneous coordinates, see §2.) For example $1 \to 2 \to 3 \to 5 \to 7 \to 9 \to 16 \to 23$ is a Lucas chain for 23.

The basic idea of Montgomery's PRAC algorithm [15] is to find a Lucas chain using some heuristics. Assume for example we want to generate $1009 \cdot P$. To generate a sequence close to optimal, a natural idea is to use as previous term $1009/\phi \approx 624$, where $\phi = (1 + \sqrt{5})/2$ is the golden ratio, but this implies to have $1009 - 624 = 385$ in the sequence. We then get $1009 \to 624 \to 385 \to 239 \to 146 \to 93 \to 53 \to 40 \to 13$. At this point we cannot continue using the same transform $(d, e) \to (e, d - e)$.

To generate $\pi \cdot P$, Montgomery starts with $(d, e) = (\pi, \lfloor \pi/\alpha \rceil)$, with $\alpha = \phi$, and iteratively uses 9 different transforms to reduce the pair (d, e), each transform using from 1 to 4 point additions or duplicates, to finally reach $d = 1$.

Montgomery improved PRAC as follows: instead of using $\alpha = \phi$ only, try several values of α, and keep that one giving the smallest cost in terms of modular multiplications. The α's are chosen

so that after a few steps, the remaining values (d, e) have a ratio near ϕ, i.e., $\alpha = (a\phi + b)/(c\phi + d)$ with small a, b, c, d. If $r = \lfloor \pi/\alpha \rceil$, the idea is to share the partial quotients different from 1 among the first and last terms from the continued fraction of π/r, hoping to have small trailing quotients.

Fig. 2 gives 10 such values of α, the first partial quotients of their continued fraction, and the total cost — in terms of curve additions or duplicates — of PRAC for all primes up to B_1, for $B_1 = 10^6$ and 10^8. For a given row, all values of α above and including this row are assumed to be used. The gain using those 10 values instead of $\alpha = \phi$ only is 3.72% for $B_1 = 10^6$, 3.74% for $B_1 = 10^8$, and the excess with respect to the lower bounds given by Theorem 8 of [15] — 2114698 for $B_1 = 10^6$ and 210717774 for 10^8 — is 3.7% and 5.1% respectively.

α	cont. frac.	$B_1 = 10^6$	$B_1 = 10^8$
$\phi \approx 1.61803398875$	$1, 1, 1, \ldots$	2278430	230143294
$(\phi + 7)/5 \approx 1.72360679775$	$1, 1, 2, 1, \ldots$	2240333	226235929
$(\phi + 2311)/1429 \approx 1.618347119656$	$1, 1, 1, 1, 1, 1, 1, 1, 2, 1, \ldots$	2226042	224761495
$(6051 - \phi)/3739 \approx 1.617914406529$	$1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, \ldots$	2217267	223859686
$(129 - \phi)/79 \approx 1.612429949509$	$1, 1, 1, 1, 1, 2, 1, \ldots$	2210706	223226409
$(\phi + 49)/31 \approx 1.632839806089$	$1, 1, 1, 1, 2, 1, \ldots$	2205612	222731604
$(\phi + 337)/209 \approx 1.620181980807$	$1, 1, 1, 1, 1, 1, 2, 1, \ldots$	2201615	222335307
$(19 - \phi)/11 \approx 1.580178728295$	$1, 1, 1, 2, 1, \ldots$	2198400	222013974
$(883 - \phi)/545 \approx 1.617214616534$	$1, 1, 1, 1, 1, 1, 1, 2, 1, \ldots$	2195552	221729046
$3 - \phi \approx 1.38196601125$	$1, 2, 1, \ldots$	2193683	221533297

Fig. 2. Total cost of PRAC with several α's, for all $\pi < B_1$ (using the best double-precision approximation of α).

3 Stage Two

Both P-1, P+1 and ECM work in an Abelian group G. For P-1, G is the multiplicative group of nonzero elements of GF(p) where p is the factor to be found; for P+1, $G = \mathrm{GF}(p^2)$; for ECM, G is an elliptic curve $E_{a,b} \bmod p$. In all cases, the calculations in G reduce to arithmetic operations — additions, subtractions, multiplications, divisions — in $\mathbb{Z}/n\mathbb{Z}$. The only computation that may fail is the inversion $1/a \bmod n$, but then a non-trivial factor of n is found, unless $a = 0 \bmod n$. A unified description of stage 2 is possible [17]; for sake of clarity, we prefer here to focus on ECM.

3.1 Overall description

Stage 1 of ECM computes a point Q on an elliptic curve E. We hope there exists a prime π in the stage 2 range $[B_1, B_2]$ such that $\pi Q = O_E \bmod p$. In such a case, while computing $\pi Q = (x : y)$ in Weierstrass coordinates[4], a non-trivial gcd will yield the prime factor p of n. A *continuation* of ECM — also called stage two, phase two, or step two — tries to find those matches. The first main idea is to avoid computing πQ using a "meet-in-the-middle" — or baby-step, giant step — strategy: instead, one computes σQ and τQ such that $\pi = \sigma \pm \tau$. If $\sigma Q = (x_\sigma : y_\sigma)$ and $\tau Q = (x_\tau : y_\tau)$, then $\sigma Q + \tau Q = 0_E \bmod p$ implies $x_\sigma = x_\tau \bmod p$. It thus suffices to compute $\gcd(x_\sigma - x_\tau, n)$ to obtain[5] the factor p.

[4] It is simpler to describe stage 2 in Weierstrass coordinates.

[5] Unless $x_\sigma = x_\tau \bmod n$ too, but if we assume x_σ and x_τ to be random, this happens with probability p/n only.

Two classes of continuations differ in the way they choose σ and τ. The *birthday paradox continuation* takes $\sigma \in S$ and $\tau \in T$, with S and T two large sets, which are either random or geometric progressions, hoping that $S+T$ covers most primes in $[B_1, B_2]$, and usually other larger primes. Brent suggest to take $T = S$.

We focus here on the *standard continuation*, which takes S and T in arithmetic progressions, and guarantees that all primes π in $[B_1, B_2]$ are hit. Assume for simplicity that $B_1 = 1$. Choose an integer $d < B_2$, then all primes up to B_2 can be written

$$\pi = i \cdot d + j, \tag{3}$$

with $S = \{i \cdot d, 0 \leq i \cdot d < B_2\}$, and $T = \{j, 0 < j < d, \gcd(j,d) = 1\}$, i.e., $\sigma = i \cdot d$ and $\tau = j$. Computing S and T costs $O(B_2/d + d)$ elliptic curve operations, which is $O(\sqrt{B_2})$ for $d \approx \sqrt{B_2}$. Choosing d with many small factors also reduces the cost. The main problem is how to evaluate all $x_\sigma - x_\tau$ for $\sigma \in S$, $\tau \in T$, and take their gcd with n.

A crucial observation is that for ECM, if $jQ = (x : y)$, then $-jQ = (x : -y)$. Thus jQ and $-jQ$ share the same x-coordinate. In other words, if one computes $x_i - x_j$ corresponding to the prime $\pi = i \cdot d + j$, one will also hit $i \cdot d - j$ — which may be prime or not — for free. This can be exploited in two ways: Either restrict to $j \leq d/2$, as proposed by Montgomery [17]; or restrict j to the "positive" residues prime to d, for example if d is divisible by 6, one can restrict to $j = 1 \bmod 6$. This is what is used in GMP-ECM.

3.2 Fast Polynomial Arithmetic

Classical implementations of the standard continuation sieve primes in $[B_1, B_2]$, and therefore require $\Theta(\pi(B_2))$ operations, assuming $B_1 \ll B_2$. The main idea of the "FFT continuation" is to use fast polynomial arithmetic to compute all $x_\sigma - x_\tau$, — or their product mod n — in less than $\pi(B_2)$ operations. It would be better to call it "fast polynomial arithmetic continuation", since any subquadratic algorithm works, not only the FFT.

Here again, two variants exist. They share the idea that what one really wants is:

$$h = \prod_{\sigma \in S} \prod_{\tau \in T} (x_\sigma - x_\tau) \bmod n, \tag{4}$$

since if any $\gcd(x_\sigma - x_\tau, n)$ is non-trivial, so will be $\gcd(h, n)$. Eq. (4) computes many $x_\sigma - x_\tau$ that do not correspond to prime values of $\tau \pm \sigma$, but the gain of using fast polynomial arithmetic largely compensates this fact.

Let $F(X)$ (respectively $G(X)$) be the polynomial whose roots are the x_τ (respectively x_σ). Both F and G can be computed in $O(M(d) \log d)$ operations over $\mathbb{Z}/n\mathbb{Z}$ with the "product tree" algorithm and fast polynomial multiplication [3, 21]. The "POLYGCD" variant interprets h as $\mathrm{Res}(F, G)$, which reduces to a polynomial gcd. It is known that the gcd of two degree-d polynomials can be computed in $O(M(d) \log d)$ too. The "POLYEVAL" variant interprets h as

$$h = \pm \prod_{\tau \in T} G(x_\tau) \bmod n,$$

thus it suffices to evaluate G at all roots of F. This problem is known as "multipoint polynomial evaluation", and can be solved in $O(M(d) \log d)$ with a "remainder tree" algorithm [3, 21].

Algorithm POLYEVAL is faster, since it admits a smaller multiplicative constant in front of the $M(d) \log d$ asymptotic complexity. However, it needs — with the current state of art — to store $\Theta(d \log d)$ coefficients in $\mathbb{Z}/n\mathbb{Z}$, instead of $O(d)$ only for POLYGCD.

Fast Polynomial Multiplication. Several algorithms are available to multiply polynomials over $(\mathbb{Z}/n\mathbb{Z})[x]$. Previous versions of GMP-ECM used Karatsuba, Toom-Cook 3-way and 4-way for polynomial multiplication, and division was performed using the Borodin-Moenck-Jebelean-Burnikel-Ziegler algorithm [9]. To multiply degree-d polynomials with the Fast Fourier Transform, we need to find $\omega \in \mathbb{Z}/n\mathbb{Z}$ such that $\omega^{d/2} = -1 \bmod n$, which is not easy, when possible.

Montgomery [18] suggests to perform several FFTs modulo small primes — chosen so that finding a primitive d-root of unity is easy — and to recover the coefficients by the Chinese Remainder Theorem. This approach was recently implemented by Dave Newman in GMP-ECM. On some processors, it is faster than the second approach described below; however, it requires to implement a polynomial arithmetic over $\mathbb{Z}/p\mathbb{Z}$, for p a small prime (typically fitting in a machine word).

The second approach uses the "Kronecker-Schönhage trick"[6]. Assume we want to multiply two polynomials $p(x)$ and $q(x)$ of degree $< d$, with coefficients $0 \leq p_i, q_i < n$. Choose $\beta^l > dn^2$, and create the integers $P = p(\beta^l)$ and $Q = q(\beta^l)$. Now multiply P and Q using fast integer arithmetic (integer FFT for example). Let $R = PQ$. The coefficients of $r(x) = p(x)q(x)$ are simply obtained by reading R as $r(\beta^l)$. Indeed, the condition $\beta^l > dn^2$ ensures that consecutive coefficients of $r(x)$ do not "overlap" in R. It just remains to reduce the coefficients modulo n.

The advantage of Kronecker-Schönhage trick is that no algorithm has to be implemented for polynomial multiplication, since one directly relies on fast integer multiplication. Division is performed in a similar way, with Barrett's algorithm: firstly multiply by the pseudo-inverse of the divisor — which is invariant here, namely $F(X)$ when using $k \geq 2$ blocks, see below —, then multiply the resulting quotient by the divisor. A factor of two can be saved in the latter multiplication, by using the "wrap-around" or "$x^d + 1$" trick, assuming the integer FFT code works modulo $2^m + 1$ [2].

3.3 Stage 2 blocks

For a given stage 2 bound B_2, computing the product and remainder trees may be relatively quite expensive. A workaround is to split stage 2 into $k > 1$ blocks [18]. Let $B_2 = kb_2$, and choose $d \approx \sqrt{b_2}$ as in §3.1. The set $S = \{i \cdot d, 0 \leq i \cdot d < b_2\}$ of §3.1 is replaced by $S_1, \ldots, S_k$ that cover all multiples of d up to B_2, which correspond to polynomials $G_1, \ldots, G_k$. The set T remains unchanged, and still corresponds to the polynomial F. Instead of evaluating G at all roots of F, one evaluates $H = G_1 G_2 \cdots G_k$ at all roots of F. Indeed, if one of the G_l vanishes at a root of F, the same holds for H. Moreover, it suffices to compute $H \bmod F$, which can be done by $k - 1$ polynomial multiplications and divisions modulo F.

Assume a product tree costs $pM(d) \log d$, and a remainder tree $rM(d) \log d$. With one unique block ($k = 1$), we compute two product trees — for F and G —, and one remainder tree, all of size d, with a total cost of $(2p + r)M(d) \log d$. With k blocks, we compute $k + 1$ product trees — for $F, G_1, \ldots, G_k$ —, and one remainder tree, all of degree about $d/\sqrt{k}$. Assuming $M(d)$ is quasi-linear, and neglecting all other costs in $O(M(d))$, the total cost is $\frac{(k+1)p+r}{\sqrt{k}} M(d) \log d$. The optimal value of k then depends on the ratio r/p. Without caching Fourier transforms, the best known ratio is $r/p = 2$ using Bernstein's "scaled remainder trees" [3]. Each node of the product tree corresponds to one product of degree l polynomials, while the corresponding node of the remainder tree corresponds to two "middle products" [4, 11]. For $r/p = 2$, the theoretical optimal value is $k = 3$, with a cost of $3.46pM(d) \log d$, instead of $4pM(d) \log d$ for $k = 1$.

[6] The idea of using this trick is due to Dave Newman; a similar algorithm is attributed to Robbins in [18, §3.4].

3.4 Brent-Suyama's Extension

Brent-Suyama's extension increases the probability of success of stage 2, with a small additional cost. Recall stage 2 succeeds when the largest factor π of the group order can be written as $\pi = \sigma \pm \tau$, where points σQ and τQ have been computed in sets S and T respectively. The idea of Brent and Suyama [5] is to compute $\sigma^e Q$ and $\tau^e Q$ instead, or more generally $f(\sigma)Q$ and $f(\tau)Q$ for some integer polynomial $f(x)$, as suggested by Montgomery [17]. If $\pi = \sigma \pm \tau$, then π divides one of $f(\sigma) \pm f(\tau)$. Thus all primes π up to B_2 will still be hit, but other larger primes may be hit too, especially if $f(x) \pm f(y)$ has many algebraic factors. This is the case for $f(x) = x^e$, but also for Dickson polynomials as suggested by Montgomery in [18]. GMP-ECM uses Dickson polynomials of parameter $\alpha = -1$ with the notation from [18]: $d_1 = 1$, $d_2 = x^2 + 2$, and $d_{e+2} = x d_{e+1} + d_e$ $d_3(x) = x^3 + 3x$, $d_4(x) = x^4 + 4x^2 + 2$.

To efficiently compute the values of $f(\sigma)Q$, we use the "table of differences" algorithm [17, §5.9]. For example, to evaluate x^3 we form the following table:

$$\begin{array}{cccccccccc} \mathbf{1} & & 8 & & 27 & & 64 & & 125 & & 216 \\ & \mathbf{7} & & 19 & & 37 & & 61 & & 91 & \\ & & \mathbf{12} & & 18 & & 24 & & 30 & & \\ & & & \mathbf{6} & & 6 & & 6 & & & \end{array}$$

Once the entries in boldface have been computed[7], one deduces the corresponding points over the elliptic curve, for example here $1Q$, $7Q$, $12Q$ and $6Q$. Then each new value of $x^e Q$ is obtained with e point additions: $1Q + 7Q = 8Q$, $7Q + 12Q = 19Q$, ... One has to switch to Weierstrass coordinates, since if iQ and jQ are in the difference table, $|i - j|Q$ is not necessarily, for example $5Q = 12Q - 7Q$ is not here. As mentioned in [18], the e point additions in the downward diagonals are performed in parallel, using Montgomery's trick to perform one modular inverse only, at the cost of $O(e)$ extra multiplications. Efficient ways to implement Brent-Suyama's trick for P-1 and P+1 are described in [17].

Note that since Brent-Suyama's extension depends on the choice of the stage 2 parameters (k, d, ...), extra-factors found may not be reproducible with other software, or even different versions of the same software.

3.5 Montgomery's $d_1 d_2$ Improvement

A further improvement is proposed by Montgomery in [17]. Instead of sieving primes in the form $\pi = id + j$ as in §3.1, use a double sieve with d_1 coprime to d_2:

$$\pi = id_1 + jd_2.$$

(The description in §3.1 corresponds to $d_1 = d$ and $d_2 = 1$.) Each $0 < \pi \leq B_2$ can be written uniquely as $\pi = id_1 + jd_2$ with $0 \leq j < d_1$: take $j = -\pi/d_2 \bmod d_1$, then $i = (\pi - jd_2)/d_1$.

To sieve all primes up to B_2, take $S = \{id_1, -d_1 d_2 < id_1 \leq B_2, \gcd(i, d_2) = 1\}$ and $T = \{jd_2, 0 \leq j < d_1, \gcd(j, d_1) = 1\}$. In comparison to §3.1: (i) the lower bound for id_1 is now $-d_1 d_2$ instead of 0, but this has little effect if $d_1 d_2 \ll B_2$; (ii) the additional condition $\gcd(i, d_2) = 1$ reduces the size of S by a relative factor $1/d_2$.

When using several blocks, the extra values of i mentioned in (i) occur for the first block only, whereas the speedup in (ii) holds for all blocks. In fact, since the size of T yields the degree of the

[7] Over the integers, and not over the elliptic curve as the author did in a first implementation!

polynomial arithmetic — i.e., $\phi(d_1)/2$ with the remark at end of §3.1 — and we want S to have the same size, this means we can enlarge the block size b_2 by a relative factor $1/d_2$ for free.

This improvement was implemented in GMP-ECM by Alexander Kruppa, up from version 6.0, with d_2 being a small prime. The following table gives for several factor sizes, the recommended stage 1 bound B_1, the corresponding effective stage 2 bound B_2', the ratio B_2'/B_1, the number k of blocks, the parameters d_1 and d_2, the degree $\phi(d_1)/2$ of polynomial arithmetic, the polynomial used for Brent-Suyama's extension, and finally the expected number of curves. All values are the default ones used by GMP-ECM 6.0.1 for the given B_1.

digits	B_1	B_2'	B_2'/B_1	k	d_1	d_2	$\phi(d_1)/2$	poly.	curves
40	$3 \cdot 10^6$	4592487916	1531	2	150150	17	14400	$d_6(x)$	2440
45	$11 \cdot 10^6$	30114149530	2738	2	371280	11	36864	$d_{12}(x)$	4590
50	$43 \cdot 10^6$	198654756318	4620	2	1021020	19	92160	$d_{12}(x)$	7771
55	$110 \cdot 10^6$	729484405666	6632	2	1891890	17	181440	$d_{30}(x)$	17899
60	$260 \cdot 10^6$	2433583302168	9360	2	3573570	19	322560	$d_{30}(x)$	43670
65	$850 \cdot 10^6$	15716618487586	18490	2	8978970	17	823680	$d_{30}(x)$	69351

As an example, with $B_1 = 3 \cdot 10^6$, the default B_2 value used for ECM is[8] $B_2 = 4592487916$ — about $1531 \cdot B_1$ —, with $k = 2$ blocks, $d_1 = 150150$, $d_2 = 17$. This corresponds to polynomial arithmetic of degree $\phi(150150)/2 = 14400$.

4 Results and Open Questions

Largest ECM factor. Records given in this section are as of January 2006. The largest prime factor found by ECM is a 66-digit factor of $3^{466} + 1$ found by Bruce Dodson on April 7, 2005:

$$p66 = 709601635082267320966424084955776789770864725643996885415676682297.$$

This factor was found using GMP-ECM, with $B_1 = 110 \cdot 10^6$ and $\sigma = 1875377824$; the corresponding group order, computed with the Magma system [14], is:

$$g = 2^2 \cdot 3 \cdot 11243 \cdot 336181 \cdot 844957 \cdot 1866679 \cdot 6062029 \cdot 7600843 \cdot 8046121 \cdot 8154571 \cdot 13153633 \cdot 249436823.$$

The largest group order factor is only about $2.3B_1$, and much smaller than the default $B_2' = 729484405666$ (see above table).

We can reproduce this lucky curve with GMP-ECM 6.0.1, here on an Opteron 250 at 2.4Ghz, with improved GMP assembly code from Torbjörn Granlund[9]:

```
GMP-ECM 6.0.1 [powered by GMP 4.1] [ECM]
Input number is 180241397103940772078159779297801504017708653303813750145082169906990204420366728928912\7\
4814402760531304131590067861951398548382931195190615371324248478807099289879585509160103851 3 (180 digits)
Using MODMULN
Using B1=110000000, B2=680270182898, polynomial Dickson(30), sigma=1875377824
Step 1 took 748990ms
B2'=729484405666 k=2 b2=364718554200 d=1891890 d2=17 dF=181440, i0=42
Expected number of curves to find a factor of n digits:
20      25      30      35      40      45      50      55      60      65
```

[8] The printed value is 4016636513, but the effective value is slightly larger, since "good" values of B_2 are sparse.

[9] Almost the same speed is obtained with Gaudry's assembly code at `http://www.loria.fr/~gaudry/mpn_AMD64/`.

```
2       4       10      34      135     617     3155    17899   111395  753110
Initializing tables of differences for F took 501ms
Computing roots of F took 29646ms
Building F from its roots took 27847ms
Computing 1/F took 13902ms
Initializing table of differences for G took 656ms
Computing roots of G took 25054ms
Building G from its roots took 27276ms
Computing roots of G took 24723ms
Building G from its roots took 27184ms
Computing G * H took 8041ms
Reducing  G * H mod F took 12035ms
Computing polyeval(F,G) took 64452ms
Step 2 took 262345ms
Expected time to find a factor of n digits:
20      25      30      35      40      45      50      55      60      65
29.45m  1.06h   2.88h   9.63h   1.58d   7.23d   36.93d  209.51d 3.57y   24.15y
********** Factor found in step 2: 709601635082267320966424084955776789770864725643996885415676682297
Found probable prime factor of 66 digits: 709601635082267320966424084955776789770864725643996885415676682297
Probable prime cofactor 254003638369639006304946260580155033416427414841076460189423633564858960970523 04\
48527170095214007673747737866527 29 has 114 digits
Report your potential champion to Richard Brent <rpb@comlab.ox.ac.uk>
(see ftp://ftp.comlab.ox.ac.uk/pub/Documents/techpapers/Richard.Brent/champs.txt)
```

Several comments can be made about this verbose output. Firstly we see that the effective stage 2 bound $B_2' = 729484405666$ is indeed larger than the "requested" one $B_2 = 680270182898$. The stage 2 parameters k, $d(= d_1)$, d_2 and the polynomial $d_{30}(x)$ are those of the 55-digit row in the above table (`dF` is the polynomial degree, and i_0 the starting index in $id_1 + jd_2$). Initializing the table of differences — i.e., computing the first downward diagonal for Brent-Suyama's extension — is clearly cheap with respect to "Computing the roots of F/G", which corresponds to the computation of the sets S and T, together with the whole table of differences. "Building F/G from its roots" corresponds to the product tree algorithm; "Computing 1/F" is the precomputation of the inverse of F for Barrett's algorithm. "Computing G * H" corresponds to the multiplication G_1G_2, and "Reducing G * H mod F" to the reduction of G_1G_2 modulo F: we clearly see the 1.5 factor announced in §3.2. "Computing polyeval(F,G)" stands for the remainder tree algorithm: the ratio with respect to the product tree is slightly larger than the announced value of 2. Finally the total stage 2 time is only 35% of the stage 1 time, for a stage 2 bound 6632 times larger!

Largest P-1 and P+1 factors. The largest prime factor found by P-1 is a 58-digit factor of $2^{2098} + 1$, found by the author on September 28, 2005 with $B_1 = 10^{10}$ and $B_2 = 13789712387045$:

$$p58 = 13720984069101393474114739782977370296495995838431646501 53,$$
$$p58 - 1 = 2^3 \cdot 3^2 \cdot 1049 \cdot 1627 \cdot 139999 \cdot 1284223 \cdot 7475317 \cdot 341342347 \cdot 2456044907 \cdot 9909876848747.$$

The largest prime factor found by P+1 is a 48-digit factor of the Lucas number $L(1849)$, found by Alexander Kruppa on March 29, 2003 with $B_1 = 10^8$ and $B_2 = 52337612087$:

$$p48 = 884764954216571039925598516362554326397028807829,$$
$$p48 + 1 = 2 \cdot 5 \cdot 19 \cdot 2141 \cdot 30983 \cdot 32443 \cdot 35963 \cdot 117833 \cdot 3063121 \cdot 80105797 \cdot 2080952771.$$

Other P-1 or P+1 factors. The author performed complete runs on the about 1000 composite numbers from the regular Cunningham table with P-1 and P+1 [22]. The largest run used $B_1 = 10^{10}$, $B_2 \approx 1.3 \cdot 10^{13}$, polynomial x^{120} for P-1, and $B_1 = 4 \cdot 10^9$, $B_2 \approx 1.0 \cdot 10^{13}$, polynomial $d_{30}(x)$ for P+1.

A total of 9 factors were found by P-1 during these runs, but strangely no factor was found by P+1. Nevertheless, the author believes that the P-1 and (especially) P+1 methods are not used enough. Indeed, if one compares the current records for ECM, P-1 and P+1, of respectively 66, 58 and 48 digits (`http://www.loria.fr/~zimmerma/records/Pminus1.html`), there is no theoretical reason why the $P \pm 1$ records would be smaller, especially if one takes into account that the $P \pm 1$ arithmetic is faster.

Largest ECM group order factor. The largest group order factor of a lucky elliptic curve is 81325590104999, for a 47-digit factor of $5^{430} + 1$ found by Bruce Dodson on December 27, 2005:

$$p47 = 29523508733582324644807542345334789774261776361,$$

with $B_1 = 260 \cdot 10^6$ and $\sigma = 610553462$; the corresponding group order is:

$$g = 2^2 \cdot 3 \cdot 13 \cdot 347 \cdot 659 \cdot 163481 \cdot 260753 \cdot 9520793 \cdot 25074457 \cdot 81325590104999.$$

This factor is a success for Brent-Suyama's extension, since the largest factor of g is much larger than B_2 (about $33.4B_2$).

From January 1st, 2000 to January 19th, 2006, a total of 619 prime factors of regular Cunningham numbers were found by ECM, P+1 or P-1 (`http://www.loria.fr/~zimmerma/ecmnet/`). Among those 619 factors, 594 were found by ECM with known B_1 and σ values. If we denote by g_1 the largest group order factor of each lucky curve, Fig. 3 shows an histogram of the ratio $\log(g_1/B_1)$. Most ECM programs use $B_2 = 100B_1$. Since $\log 100 \approx 4.6$, we see that they miss about half the factors that could be found using the FFT continuation.

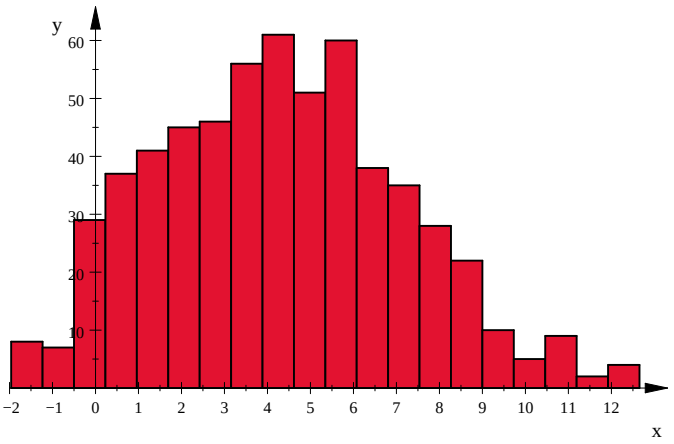

Fig. 3. Histogram of $\log(g_1/B_1)$ for 594 Cunningham factors found by ECM.

Save and Resume Interface. George Woltman's PRIME95 implementation of ECM uses the same parametrization than GMP-ECM (see §1). PRIME95 runs on x86 architectures, and factors base-2 Cunningham numbers only so far, but Stage 1 of PRIME95 is much faster than GMP-ECM, thanks to some highly-tuned assembly code. Since PRIME95 does not implement the "FFT continuation" yet, a public interface was designed to perform stage 1 with PRIME95, and stage 2 with GMP-ECM. The first factor found by this collaboration between PRIME95 and GMP-ECM was obtained by Patrik Johansson, who found a 48-digit factor of $2^{731} - 1$ on March 30th, 2003, with $B_1 = 11000000$ and $\sigma = 7706350556508580$:

$$p48 = 223192283824457474300157944531480362369858813007.$$

This save/resume interface may have other applications:

- after a stage 1 run, we may split a huge stage 2 on several computers. Indeed, GMP-ECM can be given a range $[l, h]$ as stage 2 range, meaning that all primes $l \leq \pi \leq h$ are covered. The total cpu time will be slightly larger than with a single run, due to the fact that several product/remainder trees will be computed, but the real time may be drastically decreased;
- when using $P \pm 1$, previous stage 1 runs with smaller B_1 values can be reused. If one increases B_1 by a factor of 2 after each run, a factor of 2 will be saved on each stage 1 run.

Library Interface Since version 6, GMP-ECM also includes a library, distributed under the GNU Lesser General Public License (LGPL). This library enables other applications to call ECM, P+1 or P-1 directly at the C-language level. For example, the Magma system uses the ecm library since version V2.12, released in July 2005 [14].

Open Questions. The implementation of the "FFT continuation" described here is fine for moderate-size numbers (say up to 1000 digits) but may be too expensive for large inputs, for example Fermat numbers. In that case, one might want to go back to the classical standard continuation. Montgomery proposes in [17] the PAIR algorithm to hit all primes in the stage 2 range with small sets S and T. This algorithm was recently improved by Alexander Kruppa in [12], by choosing nodes in a partial cover of a bipartite graph.

Although many improvements have been made to stage 2 in the last years, the real bottleneck remains stage 1. The main question is whether it is possible to break the sequentiality of stage 1, i.e., to get a $o(B_1)$ cost. Any speedup to stage 1 is welcome: Alexander Kruppa suggested (personal communication) to design a sliding window variant in affine coordinates. Another idea is to save one multiply per duplicate by forcing b to be small in Eq. (2); this does not work for Suyama's parametrization, since it reduces to solve a polynomial equation in σ modulo n, but maybe we can find another class of good elliptic curves parametrized by b instead of σ.

Finally, is it possible to design a "stage 3", i.e., hit two large primes in stage 2? How much would it increase the probability of finding a factor?

Acknowledgements. The author is very grateful to Richard Brent and Bruce Dodson, who encouraged him several times to write this paper. Most of the ideas described here are due to other people: many thanks of course to H. W. Lenstra, Jr., for inventing that wonderful algorithm, to Peter Montgomery and Richard Brent for their great improvements, to George Woltman who helped to design the save/resume interface, and of course to the other developers of GMP-ECM,

Alexander Kruppa, Jim Fougeron, Laurent Fousse, and Dave Newman. Part of the success of GMP-ECM is due to the GMP library, developed mainly by Torbjörn Granlund. Finally, many thanks to all users of GMP-ECM, those who found large factors as well as the anonymous users who did not (yet) found any!

References

1. Barrett, P. Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In *Advances in Cryptology, Proceedings of Crypto'86* (1987), A. M. Odlyzko, Ed., vol. 263 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 311–323.
2. Bernstein, D. Removing redundancy in high-precision Newton iteration. `http://cr.yp.to/fastnewton.html`, 2004. 13 pages.
3. Bernstein, D. J. Scaled remainder trees. `http://cr.yp.to/papers.html#scaledmod`, 2004. 8 pages.
4. Bostan, A., Lecerf, G., and Schost, E. Tellegen's principle into practice. In *Proceedings of the 2003 international symposium on Symbolic and algebraic computation* (Philadelphia, PA, USA, 2003), pp. 37–44.
5. Brent, R. P. Some integer factorization algorithms using elliptic curves. *Australian Computer Science Communications 8* (1986), 149–163. `http://web.comlab.ox.ac.uk/oucl/work/richard.brent/pub/pub102.html`.
6. Brent, R. P. Factor: an integer factorization program for the IBM PC. Tech. Rep. TR-CS-89-23, Australian National University, 1989. 7 pages. Available at `http://wwwmaths.anu.edu.au/~brent/pub/pub117.html`.
7. Brent, R. P. Factorization of the tenth Fermat number. *Mathematics of Computation 68*, 225 (1999), 429–451.
8. Brent, R. P., and Pollard, J. M. Factorization of the eighth Fermat number. *Mathematics of Computation 36* (1981), 627–630.
9. Burnikel, C., and Ziegler, J. Fast recursive division. Research Report MPI-I-98-1-022, MPI Saarbrücken, Oct. 1998.
10. Granlund, T. *GNU MP: The GNU Multiple Precision Arithmetic Library*, 4.1.4 ed., 2004. `http://www.swox.se/gmp/#DOC`.
11. Hanrot, G., Quercia, M., and Zimmermann, P. The middle product algorithm, I. Speeding up the division and square root of power series. *AAECC 14*, 6 (2004), 415–438.
12. Kruppa, A. Optimising the enhanced standard continuation of the P-1 factoring algorithm. Diplomarbeit Report, Technische Universität München, 2005. `http://home.in.tum.de/~kruppa/DA.pdf`, 55 pages.
13. Lenstra, H. W. Factoring integers with elliptic curves. *Annals of Mathematics 126* (1987), 649–673.
14. The Magma computational algebra system. `http://magma.maths.usyd.edu.au/`, July 2005. version V2.12.
15. Montgomery, P. L. Evaluating recurrences of form $x_{m+n} = f(x_m, x_n, x_{m-n})$ via Lucas chains, Dec. 1983. Available at `ftp.cwi.nl:/pub/pmontgom/Lucas.ps.gz`.
16. Montgomery, P. L. Modular multiplication without trial division. *Mathematics of Computation 44*, 170 (Apr. 1985), 519–521.
17. Montgomery, P. L. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation 48*, 177 (1987), 243–264.
18. Montgomery, P. L. *An FFT Extension of the Elliptic Curve Method of Factorization*. PhD thesis, University of California, Los Angeles, 1992. `ftp.cwi.nl:/pub/pmontgom/ucladissertation.psl.gz`.
19. Phatak, D. S., and Goff, T. Fast modular reduction for large wordlengths via one linear and one cyclic convolution. In *Proceedings of 17th IEEE Symposium on Computer Arithmetic (ARITH'17), 27-29 June 2005, Cape Cod, MA, USA* (2005), IEEE Computer Society, pp. 179–186.
20. Schönhage, A., and Strassen, V. Schnelle Multiplikation großer Zahlen. *Computing 7* (1971), 281–292.
21. von zur Gathen, J., and Gerhard, J. *Modern Computer Algebra*. Cambridge University Press, 1999.
22. Wagstaff, S. S. The Cunningham project. `http://www.cerias.purdue.edu/homes/ssw/cun/`.
23. Williams, H. C. A $p+1$ method of factoring. *Mathematics of Computation 39*, 159 (1982), 225–234.