FastECPP over MPI

Andreas Enge

FastECPP over MPI

Andreas Enge

INRIA, Université de Bordeaux, CNRS, CANARI, 33400 Talence, France
https://enge.math.u-bordeaux.fr/
andreas.enge@inria.fr

Abstract. The FastECPP algorithm is currently the fastest approach to prove the primality of general numbers, and has the additional benefit of creating certificates that can be checked independently and with a lower complexity. This article shows how by parallelising over a linear number of cores, its quartic time complexity becomes a cubic wall-clock time complexity; and it presents the algorithmic choices of the FastECPP implementation in the author's CM software
https://www.multiprecision.org/cm/
which has been written with massive parallelisation over MPI in mind, and which has been used to establish a new primality record for the "repunit" $(10^{86453}-1)/9$.

1 FastECPP and its complexity

Since its inception, ECPP has become the fastest practical algorithm for proving the primality of arbitrary numbers N; additionally, it creates a certificate that can be verified independently and in less time. It is based on the key observation that if one can find an elliptic curve over $\mathbb{Z}/N\mathbb{Z}$ and a point on the curve of sufficiently large order N', then the primality of N' implies the primality of N. One step of the certificate creation process consists of searching for a suitable value of N', smaller than N by at least one bit, the elliptic curve and the point; then it continues recursively with N'. If $L = \lceil \log_2(N) \rceil$, this requires $O(L)$ steps. Verification goes through these $O(L)$ steps again, but checking their correctness is much faster than finding them.

To find a suitable curve, Goldwasser and Kilian in their original algorithm [14] use point counting on random curves, which results in a (heuristic, under the assumption that all occurring numbers behave randomly with respect to the sizes of their prime factors) time complexity of $\tilde{O}(L^6)$ using asymptotically fast arithmetic, where the $\tilde{O}$ notation hides logarithmic factors. Atkin and Morain observe in [2] that one can find suitable elliptic curves using the complex multiplication method, which lowers the heuristic complexity to $\tilde{O}(L^5)$; this is now known as the ECPP algorithm.

The FastECPP version, attributed to Shallit in [18, §5.10] and worked out in [13, 19], improves a bottleneck, lowering the heuristic complexity to $\tilde{O}(L^4)$.

In this section, we present the FastECPP algorithm and its heuristic analysis following [19]; for a comprehensive overview of primality testing and proving

algorithms, see [20]. Basic ECPP can essentially be recovered from FastECPP by dropping substep (1) of the first phase below and by computing the square roots of the discriminants in substep (2) one by one. Since this is slower in all cases and does not even substantially simplify the implementation, the basic variant is only of historical interest.

First phase. One step of the certificate creation process determines a probable prime number N' that is smaller than N and then recursively creates a certificate for N'. Each of the steps consists of the following substeps.

(1) Write down the set $\mathcal{Q} = \{q_1^*, q_2^*, \ldots\}$ of $\tilde{\Theta}(L)$ smallest "signed primes" with $\left(\frac{q^*}{N}\right) = 1$, where $q^* = q$ if q is a prime that is 1 mod 4, $q^* = -q$ if q is a prime that is 3 mod 4, or $q^* \in \{-4, \pm 8\}$. Compute their square roots modulo N, in time $\tilde{O}(L^2)$ each or $\tilde{O}(L^3)$ altogether.
(2) Fundamental quadratic discriminants are exactly the products of signed primes (without multiplicities); create a set of $\tilde{\Theta}(L^2)$ negative discriminants D with $|D| \leqslant D_{\max} \in \tilde{O}(L^2)$ and their square roots modulo N as products of the precomputed roots (products of two elements of $\mathcal{Q}$ are enough). Try to solve the Pell equation $4N = t^2 - v^2 D$ for all D using Cornachia's algorithm, in time $\tilde{O}(L)$ per problem or $\tilde{O}(L^3)$ altogether. The success probability is of the order of $1/\sqrt{|D|} \in \tilde{\Omega}(1/L)$, so $\tilde{\Theta}(L)$ values t are expected to be obtained; then if N is prime, there are elliptic curves modulo N with complex multiplication by D and $m = N + 1 \pm t$ points.
(3) Trial factor the m as $m = cN'$, where the cofactor $c \geqslant 2$ is B-smooth for some bound B (for instance, $B = 2$), and all primes dividing N' are larger than B; each factorisation takes $\tilde{O}(L^2)$ for $B \in \tilde{O}(L^3)$ for a total of $\tilde{O}(L^3)$ (more details are given in §2).
(4) Test the N' for primality, in time $\tilde{O}(L^2)$ each for a total of $\tilde{O}(L^3)$; the expected number of remaining primes N' is in $\Theta(1)$.

So each step, of which there are $O(L)$, has a complexity of $\tilde{O}(L^3)$.

Second phase. For each of the $O(L)$ steps of the first phase, carry out the following substeps: Construct the class polynomial of degree $h \in \tilde{O}\left(\sqrt{|D|}\right)$, where h is the class number of D, in time $\tilde{O}(|D|) \subseteq \tilde{O}(L^2)$ [6]. Find a root modulo N in time $\tilde{O}(L^3)$ and write down the two corresponding CM elliptic curves; take random points on the curves in time $\tilde{O}(L^2)$ for the square roots yielding the Y-coordinates and multiply them by the cofactors c to obtain a point of prime order N'. So this phase also has $O(L)$ independent steps of complexity $\tilde{O}(L^3)$ each.

Notice that all substeps above admit a trivial parallelism of $\Theta(L)$; so using $\tilde{\Omega}(L)$ cores, the wall-clock time complexity of FastECPP becomes $\tilde{O}(L^3)$. Some of the substeps could be further parallelised to $\Theta(L^2)$ cores, but it is probably not possible to improve the complexity of a modular square root or a primality test beneath $\tilde{\Omega}(L^2)$ even in parallel.

The final certificate is of bit size $O(L^2)$ and can be verified in sequential time $\tilde{O}(L^3)$, or in wall-clock time $\tilde{O}(L^2)$ on $\tilde{\Omega}(L)$ cores.

2 Implementation choices

The only previously available free implementation of the FastECPP algorithm was written by Jared Asuncion within Pari/Gp [1] as the `primecert` function, with numbers of around 1000 digits in mind. The Cm software [5], on the other hand, encapsulated over two decades of my research on algorithms for CM of elliptic curves; so I decided to add first a sequential FastECPP implementation and eventually an MPI version tuned for massive parallelism and with record computations in mind. Both are available in Cm since version 0.4.0 as the binaries `ecpp` and `ecpp-mpi`, respectively. This section documents the choices made in the implementation and the parameter selection depending on the input size L and the available computing power. Indeed the MPI version dedicates the main process to the coordination of the computation and uses the w remaining ones to do the actual work. It adapts to the number of available cores by choosing parameters depending on w; so also the final certificate depends on w, see §5.

First phase. Substep (0). Inspired by Pari/Gp, the class numbers of all discriminants up to $D_{\max}$ are precomputed in time $\tilde{O}(D_{\max}^{1.5})$ by looping over all quadratic forms of discriminant in the desired range. Also, a number of prime products used for trial division are precomputed. Both are parallelised over all w workers, and the results can be stored on disk for reuse over several runs.

The following substeps (1) to (4) may need to be repeated over several rounds in the unlucky case that no candidate N' remains at the end.

Substep (1). Let $D_{\max} = \min\left(2^{35}, \max(2^{20}, L^2/2)\right)$ (or, more precisely, the smallest power of 2 not below this), $h_{\max} = 100000$ and $p_{\max} = \max(29, \lfloor L/2^{10} \rfloor)$. The values of $h_{\max}$ and $p_{\max}$ are intended to speed up the second phase: Only discriminants with class number at most $h_{\max}$ are considered, which is an upper bound on the degree of class polynomials constructable in reasonable time; and the maximal prime dividing the class number is bounded by $p_{\max}$, so that after expressing the class field as a tower of prime degree extensions using [8], it is enough to determine roots of polynomials of degree at most $p_{\max}$.

The main process computes the set $\mathcal{Q}$ of kw smallest signed primes and the discriminants divisible by up to 7 primes from $\mathcal{Q}$ under the additional restrictions above. The integer k is chosen minimally such that the expected number of remaining N' after substep (4), obtained using the formula for s in [13, §4], the formula in the last line of [19, §2] for the success probability of solving the Pell equation and the precomputed class numbers, is at least 3 in the first round (to allow for some choice), or 1 in later rounds (to avoid computing too many square roots). Then each worker computes k square roots modulo N.

Substep (2). The implementation evenly distributes *all* discriminants among the workers to solve the Pell equation. If the expected number of N' is much larger than 1, this may lead to unnecessary work being executed, but on the other hand, it may also lead to more choice and thus a smaller N' and ultimately fewer steps. In the absence of a clear optimality criterion, the design decision

was to favour smaller certificates over faster running times; see the comparison with Pari/Gp in §5.

Substep (3). Let P be the product of all primes up to B; then $|\log P/B - 1| \in o(1)$ by [21, Th. 4]. The numbers m_i of size $\Theta(L)$ obtained in the previous substep need to be written as $m_i = c_i N_i'$ with c_i having only prime factors dividing P, and N_i' coprime to P. The implementation follows [13, §4] and proceeds in batches to first compute all the $P \bmod m_i$. In a first step, it constructs bottom-up a binary tree with the m_i on the leaves and each inner node being the product of its two descendants; then it replaces the root $M = m_1 m_2 \cdots$ by $P \bmod M$, and top-down each node by its parent modulo the node, ending up with $P \bmod m_i$ on the leaves. We may split the m_i into batches, handled independently, such that $M \leqslant P$. Then each batch contains $O(\log_2 P/L)$ numbers m_i, and the tree computation takes time $\tilde{O}(\log P)$. If we have $\tilde{\Omega}(\log_2 P/L)$ numbers m_i, then the amortised cost per m_i is in $\tilde{O}(L)$. But we may also use larger values of B, such that $\log P \gg \log M$; the sequential version `ecpp` uses $B \in \Theta(L^3)$ for $\tilde{\Theta}(L)$ values m_i with an amortised cost of $\tilde{O}(L^2)$ per value. Now we compute the square-free part of c_i as $c_i' = \gcd(m_i, P \bmod m_i)$ in time $\tilde{O}(L)$, the square part of c_i as $c_i'' = \gcd(m_i/c_i', c_i')$ and so on. This gcd part usually has a total complexity of $\tilde{O}(L)$, but in unlikely cases (for instance if $m_i = 2^L$) may require $\tilde{O}(L^2)$. (A complexity of $\tilde{O}(L)$ could be achieved by using the algorithm of [3], which is expected to be slightly slower on average.)

Notice that the tree-based approach requires $\tilde{\Theta}(\log P)$ of memory, and that Moore's law acts similarly on the number of cores and on the memory; indeed one has observed over the last decades that the memory per core has increased only very moderately. Otherwise said, the total memory available over $\Theta(L)$ cores is of the order of $\tilde{O}(L)$, which implies that $B \in \tilde{O}(L)$: So asymptotically in the parallel setting, the effect of batch trial division vanishes.

The MPI implementation is parallelised in two dimensions, with respect to the m_i and to the primes in P. It splits the m_i into $b \approx 16$ batches, each of which is sent to an MPI communicator with $w' = \lfloor w/b \rfloor$ workers. Then worker i handles the product P_i of primes between $(i-1) \cdot 2^{29}$ and $i \cdot 2^{29}$, so that the effective smoothness bound becomes $B = w' \cdot 2^{29}$. By the already cited [21, Th. 4], each of the P_i has about the same bit size of $\log_2(P_i) \approx 2^{29}/\log(2) \approx 775 \cdot 10^6$, so that each level of the tree requires about 97 MiB of memory. In the 86453 digit record, there are up to $2^{29}/(86453 \log(10)) \approx 2700$ values m_i on the leaves, and thus $\lceil \log_2(2700) \rceil + 1 = 13$ levels, so the total memory requirements are about 1.3 GiB per core.

Substep (4). The implementation performs Miller–Rabin tests in batches of w numbers, starting with the smallest ones. So of all the non-smooth parts computed in substep (3), the smallest prime one will be retained as N'.

3 Details of the 86453 digit record

The second phase of FastECPP is built upon a number of research results that were already implemented in the CM software [5]. It computes class polynomials by complex approximations as described in [6]. Optimal class invariants are chosen derived from Weber functions [22], simple [9] or double eta quotients [10], including cases where it is enough to compute lower-degree subfields of the class field [11]. The evaluation of modular functions, which is the most important part of the class polynomial computation, is optimised following [6, 7]. To ease the step of factoring class polynomials modulo primes, the class fields are then represented as towers of cyclic Galois extensions of prime degree following [8]. The software relies on a number of libraries from the GNU project, notably GMP [15], MPFR [16] and MPC [12], and on Pari/Gp [1] for computations with class groups and Flint [17] for root finding modulo a prime.

Computations have been carried out on the PlaFRIM cluster in Bordeaux, `https://www.plafrim.fr/`, to prove primality of the "repunit" $(10^{86453}-1)/9$. The certificate is available in Pari/Gp format at

`https://www.multiprecision.org/downloads/ecpp/cert-r86453.bz2`.

An independent verification of the certificate has been carried out by the factordb server, see

`http://factordb.com/index.php?id=1100000000046752372`

The first phase computed 2980 steps in about 103 days of wall-clock time and 383 years of CPU time, in several runs with 759 to 2639 cores depending on machine availability. Of this CPU time, about 13% were devoted to the computation of square roots in substep (1), 47% to solving Pell equations in substep (2), 10% to batch trial factoring in substep (3) and 30% to primality tests in substep (4). The largest occurring discriminant in absolute value was -34223767071, the largest prime q^* was 240869 of the discriminant -3329532187, the largest prime of a class number was 277, appearing for 16 different discriminants. The effective trial factor bound $B = w' \cdot 2^{29}$ varied depending on the number w of cores assigned to a run, with $43 \leqslant w' \leqslant 172$.

The second phase was run on a machine with 96 cores and 1 TB of memory and took about 25 years of CPU time (wall-clock time was lost), that is, only about 6% of the total CPU time of both phases. Inside the phase, 2.5% of the time was devoted to computing tower representations for class fields, 2.8% to verifying orders of points on elliptic curves, and close to 95% to finding roots of class polynomials. Depending on the largest factor of the class number, which determines the degrees of the polynomials, running times for this dominant step vary considerably. The longest step took close to 42 days for factoring a class polynomial for the discriminant -2083578323 of class number $11920 = 2^4 \cdot 5 \cdot 149$ for an intermediate prime of 82089 digits; this would also have been the wall-clock time for this phase had it been run sufficiently in parallel.

Verification of the certificate took about 48 hours using Pari/Gp on the same machine with 96 cores.

4 Sizes of smooth parts

It is shown in [13, §4] that the probability of obtaining a prime quotient after removing the B-smooth part from an L-bit number is asymptotically $e^\gamma \log_2 B/L$, where $\gamma = 0.577\ldots$ is the Euler–Mascheroni constant, and that in this case the expected number of binary digits of the B-smooth part is $\log_2 B \in \tilde{O}(1)$ for B polynomial in L. The following computations give more precise estimates. They rely on approximations from analytic number theory for which very tight error bounds are available, for instance in [21]. As a consequence, the estimates are relevant for the sizes of numbers we consider, say for $B \geqslant 2^{29}$ and L of the order of a few thousands. To simplify the exposition, however, only the main terms are given, using $\sim$ to denote asymptotic equality up to lower order terms.

The probability that a number between 2^{L-1} and 2^L is a B-smooth number times a prime is given by

$$\begin{aligned}\frac{1}{2^{L-1}} \sum_{f B\text{-smooth}} \sum_{p \text{ prime},\ 2^{L-1}/f < p \leqslant 2^L/f} 1 &\sim \frac{1}{2^{L-1}} \sum_{f B\text{-smooth}} \frac{2^{L-1}}{f \log(2^L/f)} \\ &\sim \frac{1}{\log(2^L)} \sum_{f B\text{-smooth}} \frac{1}{f},\end{aligned}$$

using the prime number theorem and $\log f \in o(L)$.

For $0 < \alpha \leqslant 1$, all $f \leqslant B^\alpha$ are B-smooth, and their contribution to the sum is $\sum_{f \leqslant B^\alpha} (1/f) \sim \alpha \log B$ using the main term of the harmonic number. So the conditional probability that $f \leqslant B^\alpha$ for an L-bit number assumed to be a B-smooth number f times a prime (for L tending to infinity and B growing slowly) is α/e^γ, and the probability density function for this event with respect to α is the constant $1/e^\gamma$. In particular, $1/e^\gamma \approx 56\%$ of the primes N' remaining after substep (4) gain less than $\log_2 B$ bits compared to N.

For $1 < \alpha \leqslant 2$, numbers f with $B < f \leqslant B^\alpha$ that are *not* B-smooth are divisible by exactly one prime $P > B$, and what remains is B-smooth. So their contribution to the sum is

$$\begin{aligned}&\sum_{B < f \leqslant B^\alpha} \frac{1}{f} - \sum_{B < P \leqslant B^\alpha,\ P \text{ prime}} \frac{1}{P} \sum_{f \leqslant B^\alpha/P} \frac{1}{f} \\ &\sim (\alpha - 1)\log B - \sum_{B < P \leqslant B^\alpha,\ P \text{ prime}} \frac{1}{P}(\alpha \log B - \log P) \\ &\sim (\alpha - 1)\log B - \alpha \log B\big(\log\log(B^\alpha) - \log\log(B)\big) + \big(\log(B^\alpha) - \log(B)\big) \\ &= (2\alpha - 2 - \alpha \log \alpha)\log B\end{aligned}$$

using [21, Thm. 5 and 6]. So the probability density function is $(1 - \log\alpha)/e^\gamma$ for this range of α. In particular, $(2 - 2\log 2)/e^\gamma \approx 34\%$ of the primes N' remaining after substep (4) gain between $\log_2 B$ and $2\log_2 B$ bits compared to N, and only 10% of the numbers gain more.

For higher values of α, exact computations become unwieldy, but the previous computation still yields a *lower bound* (since non-smooth numbers with more

than one large prime factor are subtracted multiple times) on the contribution of B-smooth $f \leqslant B^\alpha$ to the sum as

$$\sum_{f \leqslant B^\alpha} \frac{1}{f} - \sum_{B < P \leqslant B^\alpha,\, P \text{ prime}} \frac{1}{P} \sum_{f \leqslant B^\alpha / P} \frac{1}{f} \sim (2\alpha - 1 - \alpha \log \alpha) \log B.$$

The function reaches its maximum $(e-1)\log B$ for $\alpha = e$. Otherwise said, the probability that an N' gains more than $2.72 \log_2 B$ bits over N is less than $1 - (e-1)/e^\gamma \leqslant 3.6\%$.

5 Comparison

As seen in the previous section, the probability that a candidate for N' gains much more than $\log_2 B$ bits is rather small, so maybe spending almost half of the time for solving Pell equations in the record to obtain more prime candidates for N' is not optimal. On the other hand, probabilities of gaining more digits in one step (and thus requiring fewer steps and less time) are amplified by the maximum statistics. For instance, the expected number of prime candidates for N' in the first step of the record was 8.9; so the probability of gaining at least $2 \log_2 B$ bits was about $1 - \big((3 - 2\log 2)/e^\gamma\big)^{8.9} \approx 58\%$, and in fact $77 = 2.2 \cdot \log_2 B$ bits were gained. Precisely, with $w = 1135$ workers, 41440753 discriminants were considered in substep (2), 40788 curve cardinalities were trial factored in substep (3), and 3405 primality tests were carried out in substep (4) before finding a prime N' (which is consistent with the expected number of 8.9 primes in a sample of 40788).

The following table compares the wall-clock times (in minutes) and the lengths of the certificates between Pari/Gp [1] (which is parallelised using threads) and Cm [5] for the first prime after 10^n for different values of n. The first two column blocks show figures for the parallel versions on a machine with 128 cores. The last column block provides the results for the serial code in Cm on the same type of machine. The initial smoothness bound B decreases (down to 2^{20}) for Pari/Gp as the N' become smaller, and remains fixed for Cm over the course of the algorithm. Notice that the storage requirement and the verification time of a certificate for a number of given size are proportional to the number of steps, so that shorter certificates are preferable.

n	Pari/Gp			Cm `ecpp-mpi`			Cm `ecpp`		
	$\log_2 B$	#steps	time	$\log_2 B$	#steps	time	$\log_2 B$	#steps	time
1000	24	131	0.30	32	33	5.2	22	88	0.80
2000	26	220	3.6	32	76	15	25	157	15
4000	30	344	41	32	166	47	28	274	210
5000	30	399	92	32	204	51	29	342	510
10000	30	740	820	32	444	220	32		

Let us first compare the parallel implementations. The Cm certificates are considerably shorter with an advantage that decreases as Pari/Gp chooses larger

smoothness bounds B. The number of steps increases roughly linearly as expected (slightly more for CM, slightly less for PARI/GP). One notices that the average gain of bits per step, computed as $\log_2(10^n)$ divided by the number of steps, is above 2.3 $\log_2 B$ for CM, illustrating the effect of the maximum statistic.

Somewhat surprisingly, neither of the two implementations shows the quartic asymptotic running time for a fixed number of cores, which may be due to "overparallelisation" and thus less than optimal CPU times for the smaller instances: CM spends 78% of the wall-clock time on trial factorisation for 5000 digits, but only 43% for 10000 digits.

The running times of the serial implementation of CM, however, reflect closely the quartic complexity; the computations for the 10000 digit number are expected to take close to a week and were not carried out. When compared to the parallel version, it also becomes apparent how the latter one profits from part of the additional computing power not for decreasing the wall-clock time, but for shortening the certificates.

The FastECPP implementation in CM has been adopted by the primality proving community, which has used it for all ECPP records since the first release of the code in CM 0.4.0 in May 2022, that is, for all but two out of the 20 entries at

`https://t5k.org/top20/page.php?id=27`.

Of the AKS/cyclotomy type competitors, the one with the best complexity known to date is the (probabilistic) algorithm in [4]. It computes an (Nd)-th power in the ring

$$\big((\mathbb{Z}/N\mathbb{Z})[Y]/(f(Y))\big)[X]/(X^e - r(Y))$$

with f of degree $d \in (\log L)^{O(\log\log\log L)} \subseteq L^{o(1)}$ and $e \in L^{2+o(1)}$, in time $L^{4+o(1)}$. While its sequential complexity is comparable to that of FastECPP, it does not seem possible to reach a wall-clock time complexity of $L^{3+o(1)}$ by parallelising it to $\Omega(L)$ cores, so that FastECPP appears to remain the algorithm of choice for proofs of large general primes in a parallel setting.

References

[1] [SW Rel.] Bill Allombert and Karim Belabas, *PARI/GP* version 2.15.4, Feb. 2024. LIC: GPL-2-or-later. URL: `https://pari.math.u-bordeaux.fr/`, SWHID: ⟨`swh:1:dir:8e76e2daa122f03e6a9206e18a62aa7ab48efb93;origin=https://pari.math.u-bordeaux`

[2] A. O. L. Atkin and F. Morain. "Elliptic Curves and Primality Proving". In: *Mathematics of Computation* 61.203 (1993), pp. 29–68.

[3] Daniel J. Bernstein. "How to find smooth parts of integers". Preprint, `https://cr.yp.to/factorization/smoothparts-20040510.pdf`. May 2004.

[4] Daniel J. Bernstein. "Proving primality in essentially quartic random time". In: *Mathematics of Computation* 76.257 (2007), pp. 389–403.

[5] [SW Rel.] Andreas Enge, *CM* version 0.4.3, Feb. 2024. LIC: GPL-3-or-later. URL: `https://www.multiprecision.org/cm/`, SWHID: ⟨`swh:1:dir:056fba450fbd9406efd86a5db93895fa63d212df;origin=https://gitlab.inria.fr/enge`

[6] Andreas Enge. "The complexity of class polynomial computation via floating point approximations". In: *Mathematics of Computation* 78.266 (2009), pp. 1089–1107.

[7] Andreas Enge, William Hart, and Fredrik Johansson. "Short addition sequences for theta functions". In: *Journal of Integer Sequences* 18.2 (2018), pp. 1–34.

[8] Andreas Enge and François Morain. "Fast Decomposition of Polynomials with Known Galois Group". In: *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes — AAECC-15*. Ed. by Marc Fossorier, Tom Høholdt, and Alain Poli. Vol. 2643. Lecture Notes in Computer Science. Extended version. Berlin: Springer-Verlag, 2003, pp. 254–264. URL: `https://enge.math.u-bordeaux.fr/publications/galois_long.ps.gz`.

[9] Andreas Enge and François Morain. "Generalised Weber Functions". In: *Acta Arithmetica* 164.4 (2014), pp. 309–341.

[10] Andreas Enge and Reinhard Schertz. "Constructing elliptic curves over finite fields using double eta-quotients". In: *Journal de Théorie des Nombres de Bordeaux* 16.3 (2004), pp. 555–568.

[11] Andreas Enge and Reinhard Schertz. "Singular values of multiple eta-quotients for ramified primes". In: *LMS Journal of Computation and Mathematics* 16 (2013), pp. 407–418.

[12] [SW Rel.] Andreas Enge et al., *GNU MPC — A library for multiprecision complex arithmetic with exact rounding* version 1.2.1, Oct. 2020. LIC: LGPL-3-or-later. URL: `https://www.multiprecision.org/mpc/`, SWHID: ⟨`swh:1:dir:ebd0a7bca44757a5e4939545d52cc68ef882a306;origin=https://gitlab.inria.fr/mpc/`

[13] J. Franke et al. "Proving the Primality of Very Large Numbers with fastECPP". In: *Algorithmic Number Theory — ANTS-VI*. Ed. by Duncan Buell. Vol. 3076. Lecture Notes in Computer Science. Berlin: Springer-Verlag, 2004, pp. 194–207.

[14] Shafi Goldwasser and Joe Kilian. "Almost All Primes Can be Quickly Certified". In: *Proc. 18th Annual ACM Symp. on Theory of Computing*. 1986, pp. 316–329.

[15] [SW Rel.] Torbjörn Granlund et al., *GMP — The GNU Multiple Precision Arithmetic Library* version 6.2.1, Nov. 2020. LIC: LGPL-3-or-later. URL: `https://gmplib.org/`, SWHID: ⟨`swh:1:dir:31da2a73b2e10e765fb52996d15e6f5f453453a3;origin=https://gmplib.org/repo/gmp-`

[16] [SW Rel.] Guillaume Hanrot et al., *GNU MPFR — A library for multiple-precision floating-point computations with exact rounding* version 4.1.0, July 2020. LIC: LGPL-3-or-later. URL: `https://www.mpfr.org/`, SWHID: ⟨`swh:1:dir:0e32d50b65ab886c5bcd44f63e3394980ad2fcdb;origin=https://gitlab.inria.fr/mpfr`

[17] [SW Rel.] William Hart et al., *FLINT: Fast Library for Number Theory* version 2.9.0, June 2022. LIC: LGPL-

2.1-or-later. URL: https://flintlib.org/, SWHID: ⟨swh:1:dir:d7c1dec3fb591a70205462058c842a245c68dd31;origin=https://github.com/flintlib/
[18] A. K. Lenstra and H. W. Lenstra Jr. "Algorithms in Number Theory". In: *Algorithms and Complexity*. Ed. by Jan van Leeuwen. Vol. A. Handbook of Theoretical Computer Science. Amsterdam: Elsevier, 1990, pp. 674–715.
[19] F. Morain. "Implementing the asymptotically fast version of the elliptic curve primality proving algorithm". In: *Mathematics of Computation* 76.257 (2007), pp. 493–505.
[20] François Morain. "La primalité en temps polynomial [d'après Adleman, Huang; Agrawal, Kayal, Saxena]". In: *Astérisque* 294.917 (2004), pp. 205–230.
[21] J. Barkley Rosser and Lowell Schoenfeld. "Approximate Formulas for Some Functions of Prime Numbers". In: *Illinois Journal of Mathematics* 6 (1962), pp. 64–94.
[22] Reinhard Schertz. "Die singulären Werte der Weberschen Funktionen $\mathfrak{f}$, $\mathfrak{f}_1$, $\mathfrak{f}_2$, γ_2, γ_3". In: *Journal für die reine und angewandte Mathematik* 286/287 (1976), pp. 46–74.