Square root algorithms for the number field sieve

Emmanuel Thomé

Square Root Algorithms for the Number Field Sieve

Emmanuel Thomé

INRIA Nancy, Villers-lès-Nancy, France

Abstract. We review several methods for the square root step of the Number Field Sieve, and present an original one, based on the Chinese Remainder Theorem.

We consider in this article the final step of the Number Field Sieve (NFS) factoring algorithm [3], namely the algebraic square root computation. This problem is stated as follows. Let $K = \mathbb{Q}(\alpha)$ be a number field, where α is defined as a root of the irreducible polyomial $f(x) \in \mathbb{Z}[x]$. We further use the notation $d = \deg f = [K : \mathbb{Q}]$, and denote by $\mathcal{O}_K$ the ring of integers of K. For brevity, we assume throughout this article that f is a monic polynomial, although the more general case $f_d \neq 1$ can be treated similarly by considering $f_d\alpha$ in lieu of α. Let $\mathcal{S}$ be a set of pairs (a, b) such that $S(\alpha) = \prod_{(a,b)\in\mathcal{S}}(a - b\alpha)$ is known to be the square of an algebraic integer. Such a set $\mathcal{S}$ is the outcome of the linear algebra and character steps of the Number Field Sieve. The purpose of the algebraic square root step is the computation of a polynomial $T(x) \in \mathbb{Z}[x]$ such that $T(\alpha)^2 = f'(\alpha)^2 S(\alpha)$. Here, $f'(\alpha)^2$ is introduced merely in order to take into account the possibility that a square root for $S(\alpha)$, despite being an algebraic integer, needs not belong to the order $\mathbb{Z}[\alpha]$. For brevity again, this $f'(\alpha)^2$ term will be omitted throughout this article.

Two further characteristics of the problem, specifically related to the NFS context, are also important. The ideals $(a - b\alpha)\mathcal{O}_K$ always have a known factorization into prime ideals (the latter being for example readily available alongside with the pairs (a, b), e.g. in a file). Furthermore, the output defined as $T(x)$ above is not interesting *per se*. In the NFS, one intends to compute $T(m) \bmod N$, where m and N are known (N being the integer to factor). In some cases, it is possible to achieve this goal without explicitly computing $T(x)$.

This article reviews several approaches for the algebraic square root task. Most of them are classical, and date back to the early research on the NFS. Let us recall that back in 1993, this square root step was regarded as difficult. Attacking the problem "directly" by computing the algebraic number $S(\alpha)$ and later its square root, appeared a daunting task by then, and this justified the development of ad hoc algorithms, such as Montgomery's [14, 15] or Couveignes' [5] algorithms, which exploit the fact that the factorization of the ideals $(a - b\alpha)\mathcal{O}_K$ is known. The practicality of asymptotically fast methods appeared later and the direct approach then became realistic. Furthermore, while slower than e.g. Montgomery's algorithm, this method has turned out to be acceptably

fast, notably when compared with the overall cost of NFS. Because the direct approach embarks less algebraic number theory background than Montgomery's method, it is sometimes preferred in NFS implementations [17, 8]. However the direct approach, when stated in its simple form, is unable to tackle square root problems arising with number fields having no inert primes (e.g. with Galois group $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z}$). Such fields are typically encountered with the Special Number Field Sieve, and provide one of the justifications for the Chinese Remainder Theorem (CRT)-based approach presented in this article.

The algorithms presented in this article are all relevant, when applicable, for modern integer factoring, and various working implementations can be found in publicly available software [8, 17, 13]. These different algorithms have actually been used or at least tested for the `rsa768` factoring effort [11]. Furthermore, in the context of the oracle-assisted RSA problem addressed in [10], a variant of Montgomery's algorithm is used, and proves particularly well-adapted.

This article is organized as follows. The direct approach for computing the square root is described in Section 1. Couveignes' CRT-based method for the odd-degree case is described in Section 2. Section 3 describes Montgomery's algorithm. The new CRT-based approach we propose is presented in Section 4. Some viable approaches for the square root computation are not detailed here (e.g. the one proposed in [4, § 3.6.2]), since they offer no obvious advantage over the ones presented here.

Complexities of the algorithms presented are given as functions of the input size. In order to be able to provide comparable estimates, we afford some simplifying assumptions. We assume that the set of pairs $\mathcal{S}$ occupies n bits. This leads to coefficients of $S(\alpha)$ each having size roughly n bits as well. The bit size of $S(\alpha)$ is thus $O(dn)$ overall. Similarly, coefficients of $T(\alpha)$ have size roughly $n/2$ bits. We acknowledge that these estimates are slightly gross, but these do make sense for the NFS context, and hold in practice. Furthermore, the number field is considered constant, so that most dependencies on the number field parameters are deliberately ignored. The notation $\mathsf{M}(k)$ is used throughout the document to denote the time for multiplying k-bit integers.

1 The direct (lifting) approach

The direct method, which is also referred to as the p-adic, or lifting approach, applies when there exists an *inert* prime in the number field K. This is only possible when the Galois group of the polynomial f admits an element of order d. Generically, the Galois group of a polynomial used in the General Number Field Sieve (GNFS) algorithm is the full symmetric group $\mathfrak{S}_d$, which implies that this property is expected to be satisfied with overwhelming probability. This is not so, however, with polynomials considered in the context of the Special Number Field Sieve (SNFS). There, polynomials typically have some special shape, which makes all sorts of Galois groups plausible. For instance, a degree 4 polynomial with Galois group $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z}$ can be encountered. This very case is

too frequent to be completely neglected, which makes the direct approach only a partial solution to our problem.

1.1 Working p-adically

Let thus p be an inert prime (such that $p\mathcal{O}_K$ is a prime ideal). To fix ideas, p is taken of manageable size (say at most 64 bits). The extension $K_p = \mathbb{Q}_p/f(x)$ is thus a degree d unramified extension of $\mathbb{Q}_p$. Let α_p be a root of f in K_p. We have a natural injective ring morphism from $\mathbb{Z}[\alpha]$ to $\mathbb{Z}_p[\alpha_p]$. The purpose of the lifting approach is to use this injective morphism in order to recognize the square root being sought.

The first step of the algorithm is the computation of a low-precision square root for $S(\alpha_p)$. Let $\mathbb{F}_p(\beta) = \mathbb{F}_{p^d}$ be the residue field of K_p, the projection $K_p \to \mathbb{F}_{p^d}$ being given by $\alpha_p \mapsto \beta$. By low-precision, we understand the computation of a square root of $S(\beta) \in \mathbb{F}_{p^d}$. Since $S(\beta) = \prod_{(a,b)\in\mathcal{S}}(a - b\beta)$ and all (a, b)'s are coprime, we know that $S(\beta) \neq 0$. Let $T(x) \in \mathbb{Z}[x]$ be, as above, a polynomial defining a square root $T(\alpha)$ of $S(\alpha)$. We know that $T(\beta)^2 = S(\beta)$, and our computation in $\mathbb{F}_{p^d}$ gives us the coefficients of $T(x)$ (or its opposite) modulo p. In the field K_p, we have

$$T(\alpha_p)^2 \in S(\alpha_p) + p\mathcal{O}_K.$$

We thus have low-precision knowledge on the coefficients of T, together with a defining equation. A Newton lifting approach then allows to recover all coefficients. For example, iterating the modification $T(\alpha_p) \leftarrow T(\alpha_p) + \frac{S(\alpha_p)-T^2(\alpha_p)}{2T(\alpha_p)}$ is sufficient. In practice, it is desirable to first compute the inverse square root instead, so that the iteration avoids inverse computations. Details are skipped, and can be found in [1, 2].

The key concern is the determination of the stopping point of the lifting process. Since coefficients of the desired solutions are known to be integers, we know that above a certain lifting step, the p-adic coefficients obtained for $T(x)$ no longer evolve. (The morphism $\mathbb{Z}[\alpha] \to \mathbb{Z}[\alpha_p]$ being injective, these coefficients are then the desired ones.) Therefore the number of lifting steps is controlled by a bound on the result coefficients, which can be provided with classical tools as we do now.

1.2 Bound on the square root coefficients

Given a number field K, we denote by $\sigma_1, \ldots, \sigma_r$ and $\sigma_{r+1}, \ldots, \sigma_{r+s}$ the real and non-conjugate complex embeddings. We further denote by $\sigma_{r+s+k} = \overline{\sigma_{r+k}}$ for $k = 1, \ldots, s$. Let $U = \sum_{i=0}^{d-1} u_i\alpha^i \in K$. The coefficients u_i of U are related to the embedding values, since these are given by a polynomial expression with coefficients u_i. Namely, we have

$$(\sigma_1(U), \ldots, \sigma_d(U)) = (u_0, \ldots, u_{d-1}) \times V\,(\sigma_1(\alpha), \ldots, \sigma_d(\alpha))$$

where the matrix above is of Vandermonde type. Exploiting this relation in the opposite direction, it is possible to derive a bound on the $|u_i|$ from a bound on the $|\sigma_i(U)|$. A low precision computation of the inverse of the Vandermonde matrix above suffices to obtain a reasonable bound at a moderate cost.

Such a mechanism is typically used to bound the coefficients of an element U given the *logarithms* of the embeddings, where logarithms provide an additional guard against exponent overflow. This can precisely be done in the context of NFS square root computations. We know $S(\alpha)$, and look for $T(\alpha) \in \mathbb{Z}[\alpha]$ with $T(\alpha)^2 = S(\alpha)$. Hence we have $\log|\sigma_i(T(\alpha))| = \frac{1}{2}\log|\sigma_i(S(\alpha))|$. Furthermore, $S(\alpha)$ is also known in *product* form. The computation

$$\log|\sigma_i(S(\alpha))| = \sum_{(a,b)\in\mathcal{S}} \log|\sigma_i(a - b\alpha)|$$

therefore appears rather accessible, as its computational cost is essentially that of reading the input (set of (a, b) pairs).

The computation of the required number of lifting step therefore proceeds as follows.

- Compute the complex roots of f. A computation with limited precision suffices. The inverse of the matrix $V(\sigma_1(\alpha), \ldots, \sigma_d(\alpha))$ is then computed (this is a Lagrange interpolation matrix). It is important, in order for the bound to be valid, to use rounding towards $+\infty$ in the computations.
- Then, the $\log|\sigma_i(S(\alpha))|$ values may be computed.
- Given this data, deduce a bound on the coefficients $|t_i|$.

Given a bound M obtained by this method, lifting may stop at precision $k = \lceil \log_p M \rceil$. This implies in particular that in computations where $S(\alpha)$ appears, coefficients may be reduced modulo p^k, which provides noticeable savings in computation time.

One may notice that this bound computation requires no memory.

1.3 Complexity

The computation of a square root by Newton lifting is quasi-linear [1,2]. This also holds in our case of interest here. We may write the complexity as $O(d^2\mathsf{M}(n))$, where n is the size in bits of the input coefficients, and where d^2 is taken for the cost of multiplying polynomials of degree $d-1$. (One may of course use better algorithms than the naive one for this task. This is relevant only to a limited extent in our range of interest, since d denotes an NFS degree.) Note that this complexity is in fact dominated by the complexity of the preliminary computation of $S(\alpha)$ from the set $\mathcal{S}$, which claims $O(d^2\mathsf{M}(n)\log n)$ using a subproduct tree.

The space complexity of the direct approach is linear, namely $O(dn)$. In comparison to other approaches considered in this article, this approach does compute $T(x)$ as a prerequisite before computing $T(m)$ as required by NFS.

2 Couveignes' algorithm

In [5], Couveignes proposes an algorithm which allows to avoid the space complexity of the direct method above, and allows some parallelism. This approach is based on the Chinese Remainder Theorem (CRT), and is only applicable under the simultaneous conditions that the number field degree d be odd, and that there exist inert primes[1].

The approach goes as follows. We intend to compute $T(\alpha)$ with a CRT approach. Let $\{p_i\}$ be a collection of inert primes. For each such p_i, we denote by β_i a root of $f(x)$ modulo p_i, and consider (often implicitly) the ring morphism from K to $\mathbb{F}_{p_i^d}$ sending α to β_i. We begin by computing a square root of $S(\beta_i)$ for all i. Such a square root is denoted by $T_i'(\beta_i)$, and writes as $T_i'(\beta_i) = \pm T(\beta_i)$. We then wish to reconstruct the result $T(x)$ as a polynomial with integer coefficients. For this purpose, we assume that a bound M on the coefficients of T has been obtained in a manner similar to the approach described in Subsection 1.2. We assume that the product of the chosen primes p_i exceeds the bound M. Suppose then that $T_i(x) = T(x) \bmod p_i$ is known. Let q_i be the smallest positive integer such that $q_i \bmod p_j = \delta_{i,j}$ (one may write $q_i = s \cdot (s^{-1} \bmod p_i)$, where $s = \prod_{j \neq i} p_j$). We have then:

$$T(x) = \sum_i q_i T_i(x).$$

The stumbling block for such an approach is related to the choice of roots. We have acquired knowledge of $\pm T(\beta_i)$, but this is only sufficient to determine $T_i(x)$ up to a sign. In presence of a large number of primes p_i, finding the correct sign combination in the expression $\sum_i \pm q_i T_i'(x)$ is intractable.

In order to overcome this problem, Couveignes' algorithm takes advantage of the degree of K being odd. Under this assumption, we have

$$\mathrm{Norm}_{K/\mathbb{Q}}(-\zeta) = -\mathrm{Norm}_{K/\mathbb{Q}}(\zeta)$$

for any $\zeta \in K$. In the particular case of the NFS square root, computing the absolute value $|\mathrm{Norm}_{K/\mathbb{Q}}(T(\alpha))|$ is rather easy. Indeed, we know the factored form of the principal ideal $S(\alpha)\mathcal{O}_K$. We may thus write:

$$\begin{aligned} S(\alpha)\mathcal{O}_K &= \prod_{\mathfrak{p} \in \mathcal{F}} \mathfrak{p}^{2e_\mathfrak{p}} \text{ with } e_i \in \mathbb{Z}, \\ |\mathrm{Norm}_{K/\mathbb{Q}}(S(\alpha))| &= \prod_{\mathfrak{p} \in \mathcal{F}} \mathrm{Norm}(\mathfrak{p})^{2e_\mathfrak{p}}, \\ \mathrm{Norm}_{K/\mathbb{Q}}(T(\alpha)) &= \pm \prod_{\mathfrak{p} \in \mathcal{F}} \mathrm{Norm}(\mathfrak{p})^{e_\mathfrak{p}}. \end{aligned}$$

[1] The existence of one or many inert primes are equivalent conditions, in virtue of Čebotarev's density theorem. This theorem appears in countless graduate level algebraic number theory textbooks. For an introduction to Čebotarev's density theorem, including also historical aspects and applications, the article [12] is an interesting read.

We acknowledge the fact that our chosen notation $T(x)$ for the solution being sought is ambiguous, given that there are two solutions. This accounts for the $\pm$ sign in the equation above. From now on, we intend to focus on the computation of one of these two solutions only, and we need to make one single consistent choice. To this end, we require that the computed $T(x)$ correspond to a positive norm above. Such an arbitrary choice is legitimate, provided it is done only once (if we were to combine mixed information related to either of the two different solutions for many primes, consistent reconstruction would be impossible). Let thus ν be this positive norm, which is accessible to calculation. We have $\mathrm{Norm}_{K/\mathbb{Q}}(T(\alpha)) = \nu$ and $\mathrm{Norm}_{K/\mathbb{Q}}(-T(\alpha)) = -\nu$. Modulo p_i, this property transfers conveniently. We have

$$\mathrm{Norm}_{\mathbb{F}_{p_i^d}/\mathbb{F}_{p_i}}(T(\beta_i)) = \nu \bmod p_i.$$

This implies that given $T'_i(\beta_i)$, it is possible to decide whether $T'_i(\beta_i) = T_i(\beta_i)$ or $T'_i(\beta_i) = -T_i(\beta_i)$, by comparing $\mathrm{Norm}_{\mathbb{F}_{p_i^d}/\mathbb{F}_{p_i}}(T'_i(\beta_i))$ with $\nu \bmod p_i$.

Once the sign problem has been solved, it is possible to use the expression $T(x) = \sum_i q_i T_i(x)$. Note though that for the NFS application, only the quantity $T(m) \bmod N$ is needed eventually. Therefore, computing $T(x)$ is unnecessary. It is sufficient (and considerably cheaper) to write

$$T(m) \bmod N \equiv \sum_i (q_i \bmod N)(T_i(m) \bmod N).$$

Complexity Let us assume that primes of fixed size λ are considered[2] Since the result coefficients occupy $O(n)$ bits, it suffices to consider $O(n/\lambda)$ primes. We first consider the complexity of such an approach in the perspective of a constant space complexity. For each such prime, one has to read the input set $\mathcal{S}$, in order to compute the norm ν. This step dominates the complexity, since it takes $O(n^2)$. Furthermore, parallelizing this step is not necessarily obvious, since the complexity essentially consists of input-output operations (several threads of a single processor may benefit from a single read at the same time, but such a benefit does not scale well to a multi-machine setup).

This approach is amenable to a time-memory trade-off. It is possible to read the set $\mathcal{S}$ in $\sqrt{n}$ blocks of size $\sqrt{n}$. For each such block, an intermediary product of size $\sqrt{n}$ may be computed, and then reduced modulo each prime p_i. In such a way, the time complexity drops to $O(n^{3/2})$, for a space complexity in $O(n^{1/2})$. Such a modification could be considered in a distributed setting, as it offers more opportunities for parallelization.

3 Montgomery's algorithm

Montgomery's algorithm [14, 15] for the square root step is the one which is most specially crafted for the NFS square root situation. This algorithm radically

[2] The obvious finiteness of the number of primes satisfying this criterion is an irrelevant concern here.

differs from the two algorithms presented above, as well as from the one which is described in Section 4. The original description of Montgomery's algorithm is given in [14], although the unpublished draft [15] is also worth reading in that it contains additional details. Nguyen also presented Montgomery's algorithm in [16].

Montgomery's algorithm is not dependent on strong assumptions as the algorithms presented above. In particular, the Galois group of the defining polynomial is not an obstacle for Montgomery's algorithm. However, the knowledge of the factorization of the ideal $S(\alpha)\mathcal{O}_K$ as a product involving only ideals of small norm is crucially important. Furthermore, as we will see, the "dependency on 2" is especially good with this algorithm. More precisely, this algorithm may be stated in a more general way as an algorithm for computing a λ-th root, where λ is any integer, provided that the factorization of $S(\alpha)$ remains known. Of course, in such a case, one expects the coefficients of $T(\alpha)$ to be λ times smaller than those of $S(\alpha)$. This generalization matters to the oracle-assisted computation of λ-th roots as detailed in [10], and accounts for our choice to present Montgomery's algorithm in the extended setting $\lambda \geq 2$.

We are interested in the factored form of the ideal $S(\alpha)\mathcal{O}_K$. In the NFS context, the computation of the maximal order $\mathcal{O}_K$ of the number field K is a hard problem, because the discriminant of polynomials used in the (general) NFS is possibly even harder to factor than the number which we intend to factor in the first place. Yet, given a monic polynomial $f(x)$, factoring its discriminant is necessary in order to decide which are the primes p for which the index $[\mathcal{O}_K : \mathbb{Z}[\alpha]]$ is divisible by p. For such primes, we say that *locally at* p, the order $\mathbb{Z}[\alpha]$ is not maximal, since $\mathcal{O}_K$ is larger.

A very important observation is that for the purpose of factoring ideals over a prescribed factor base, which consists of all primes ideals above a prescribed set of primes, it is sufficient to work with an order $\mathcal{O}$ which is maximal at those primes only, and needs not be maximal over all primes. For the ideals considered, factorization into $\mathcal{O}$-ideals or $\mathcal{O}_K$-ideals coincide. Extension of a starting order to a p-maximal one for a finite set of primes p can be done using Zassenhaus' Round-2 algorithm, as presented for example in [4]. The computation of such an order $\mathcal{O}$ is an inexpensive preliminary step for Montgomery's algorithm, and we assume it is done.

The following paragraphs use the notation $\mathcal{F}$ for the set of prime ideals occuring in the factorization of $S(\alpha)$ (considerations related to "large primes" aside, this can be thought of as the factor base).

3.1 Iterative reduction

The main idea of the algorithm is the following. The computation has an iterative structure, and the successive steps are numbered from step number 0 onwards. Throughout the course of the computation, an expression such as the following one is maintained (we recall that λ denotes the order of the root which we intend

to compute, e.g. classically $\lambda = 2$):

$$S(\alpha)(\gamma_0^{\epsilon_0} \dots \gamma_{k-1}^{\epsilon_{k-1}})^{-\lambda}\mathcal{O} = \prod_{\mathfrak{p}\in\mathcal{F}} \mathfrak{p}^{\lambda \cdot e_{\mathfrak{p}}^{(k)}}$$

where the integer k denotes the step of the calculation, starting at $k = 0$. We use γ_i to denote some algebraic number which is computed at step i. We also denote by ϵ_i a sign, which is used as a notational convenience for a choice between numerator and denominator. The exponents $e_{\mathfrak{p}}^{(k)}$ may become negative in the course of the computation.

The aim of these reduction steps is to transform the left-hand side above into an algebraic number which has small coefficients. Not only do we wish to obtain an ideal factorization which is as small as possible (possibly trivial), but we also strive to minimize the *unit contribution* as well, which is important to minimize the size of the coefficients.

We denote by $S_k(\alpha) = S(\alpha)(\gamma_0^{\epsilon_0} \dots \gamma_{k-1}^{\epsilon_{k-1}})^{-\lambda}$ (in particular, $S_0(\alpha) = S(\alpha)$). Step k of the algorithm chooses a subset of the ideals appearing in the factorization $\prod_{\mathfrak{p}\in\mathcal{F}} \mathfrak{p}^{\cdot e_{\mathfrak{p}}^{(k)}}$. This factorization naturally decomposes into a numerator (positive exponents) and a denominator (negative exponents). We choose a set $I_k = \mathfrak{p}_1 \dots \mathfrak{p}_{n_k}$, which consists of ideals all appearing in the numerator, or all in the denominator (in this notation we do not forbid a given ideal to be selected several times, provided that its multiplicity in I_k does not exceed $|e_{\mathfrak{p}}^{(k)}|$). We intend to reduce the contribution of the the ideals $\mathfrak{p}_1, \dots, \mathfrak{p}_{n_k}$ to the factorization of $S_k(\alpha)\mathcal{O}$.

A reduced basis (e.g., an LLL-reduced basis suffices) of the ideal I_k is computed. In this way, we obtain algebraic integers belonging to I_k. Such an algebraic integer v satisfies $\mathrm{Norm}_{K/\mathbb{Q}}(I_k) \mid \mathrm{Norm}_{K/\mathbb{Q}}(v)$, and also the following inequality:

$$m(v) \stackrel{\text{def}}{=} \left| \frac{\mathrm{Norm}_{K/\mathbb{Q}}(v)}{\mathrm{Norm}_{K/\mathbb{Q}}(I_k)} \right| \leq C_K,$$

where the constant C_K is effectively computable (it depends only on K). Suppose we are given such an element v. Without loss of generality, we consider the case that I_k is a factor of the numerator of $S_k(\alpha)\mathcal{O}$. We then have:

$$\mathrm{Norm}(S_k(\alpha)v^{-\lambda}) = \mathrm{Norm}(S_k(\alpha))\,\mathrm{Norm}(I_k)^{-\lambda} m(v)^{-\lambda}.$$

In the factorization of $S_k(\alpha)v^{-\lambda}\mathcal{O}_K$, the norm of the numerator is reduced by a factor $\mathrm{Norm}(I_k)^{\lambda}$ in comparison to $S_k(\alpha)\mathcal{O}_K$. At the same time, the norm of the denominator increases by $m(v)^{\lambda} \leq C_k^{\lambda}$. Therefore, provided that I_k is chosen to have a norm significantly larger than the constant C_K, we obtain in this way a reduction of the size of the expression. This reduction step is the workhorse of Montgomery's algorithm.

The iterative reduction step may possibly complete with a trivial ideal factorization, in other words with some $S_k(\alpha)$ being a unit. For this unit to be acceptably small, and accessible to direct λ-th root computation, it is important

to "guide" the reduction step. The guiding principle is that the logarithms of the complex embeddings $\log|\sigma_i(S_k(\alpha)v^{-\lambda})|$ should become as balanced as possible: among the different short vectors v formed by the reduced basis of I_k, some lead $S_k(\alpha)v^{-\lambda}\mathcal{O}_K$ to reduce the jitter in the logarithmic complex embeddings, while some others have the opposite effect. The former are favored. In practice this suffices to obtain a final $S_k(\alpha)$ which is a unit (thus the ideal factorization is indeed trivial), and furthermore has trivial logarithmic embeddings, so that it is actually a root of unity.

More details on Montgomery's algorithm, together with an example, can be found in [15]. An implementation of Montgomery's algorithm can be found in [13]. We also mention a Magma prototype by the author in [8].

3.2 Complexity

Complexity of Montgomery's algorithm, especially in the extended case $\lambda \geq 2$ which is useful for the context in [10], calls for a more precise consideration of the input and output sizes. We assume that the input set $\mathcal{S}$ occupies n bits. This set $\mathcal{S}$ is obtained in a manner similar to the NFS situation: the solution of some linear system is computed so as to force exponents in the corresponding ideal factorization to cancel modulo λ. Naturally, this linear system is defined modulo λ, and so are the coefficients of this combination. Therefore, our set $\mathcal{S}$ naturally consists of (a, b) pairs together with exponents for each pair, which contribute to the size of $S(\alpha)$. For an input size of n bits for $\mathcal{S}$, we thus expect $O(\lambda n)$ bits for the coefficients of $S(\alpha)$, and $O(n)$ bits for the coefficients of its λ-th root $T(\alpha)$.

Observe now that $S(\alpha)$ is never computed by Montgomery's algorithm. Each reduction step has a cost which is linear in the size of I_k, and more importantly in the size reduction obtained with respect to the input set. The calculations related to the unit contributions are also done incrementally, and have linear cost in the size of I_k. Therefore the complexity of the algorithm is linear, and the dependency on λ is of logarithmic type. The dependency on the parameters of the field K is not detailed here.

4 A new CRT-based lifting approach

We describe here a new approach which is a mix between the direct approach (Section 1) and Couveignes' CRT algorithm (Section 2). This approach, just as Montgomery's algorithm, is free of limiting assumptions on the number field. In particular, we do not assume the existence of inert primes.

This work may be considered as connected to recent works by Enge and Sutherland [6], as well as by Sutherland [18], on the topic of the CRT-based computation of class polynomials in the context of the complex multiplication method for constructing elliptic curves over finite fields. In these works, as well as in the method described here, the "explicit" aspect of the CRT is particularly important.

Let $d = [K : \mathbb{Q}]$. We recall that for brevity we have restricted our presentation to the case where the polynomial f defining the number field K is monic. As stated already before, we know a way to obtain a bound on the coefficients of the result $T(\alpha)$ which is being sought. We thus assume that such a bound M has been computed and is available.

Let $\mathcal{P} = \{p_i\}$ be a set of $\ell = t \times r$ totally split primes[3]. (The form $\ell = t \times r$ will allow us later to partition $\mathcal{P}$ into t distinct subsets of size r, for distribution purposes.) Let $P = \prod_{p \in \mathcal{P}} p$ and $\lambda = \left\lceil \frac{\log M/\epsilon}{\log P} \right\rceil$, where $\epsilon \leq 1$ is an arbitrarily chosen value discussed later. We thus have $M \leq \epsilon P^\lambda$. We denote by $B = \frac{\lambda}{\ell} \log_2 P$ the bit-size of p^λ for primes $p \in \mathcal{P}$.

Following our assumptions for estimating complexities, the expected bit size of the coefficients of $T(\alpha)$ is $n/2$, where as previously n denotes the size of the input set $\mathcal{S}$. We thus expect expect $\log_2 M \approx \ell B \approx n/2$.

4.1 CRT-based reconstruction

The square root is computed from several calculations done modulo p_i^λ, for each p_i. One of the difficulties is naturally linked to resolving the indetermination among the two possible choices of square roots in each of the possible residue fields $\mathbb{F}_{p_i}$. In total, the primes p_i being totally split, we have ℓd square roots to compute, and as many choices to make.

For each p_i, we denote by $(r_{i,j})_{j=1\ldots d}$ the roots of f modulo p_i. Since p_i is totally split, the values $r_{i,j}$ are d distinct elements of $\mathbb{F}_{p_i}$. This assumption also allows to compute, corresponding to each $r_{i,j}$, a lift $\tilde{r}_{i,j}$ in $\mathbb{Z}/p_i^\lambda\mathbb{Z}$. We thus have $f(\tilde{r}_{i,j}) \equiv 0 \mod p_i^\lambda$.

For each p_i and each $r_{i,j}$, we compute a p_i-adic lift of $\sqrt{S(\tilde{r}_{i,j})}$, with precision λ. Let $T'_{i,j}$ be this lift. If $T(x)$ denotes as usual the expression of our desired square root in K, we have

$$T'_{i,j} = s_{i,j} T(\tilde{r}_{i,j}) \quad \text{where } s_{i,j} = \pm 1.$$

We wish to find $T(x)$ from the values $T(\tilde{r}_{i,j})$. The latter are, for now, known only up to the sign $s_{i,j}$, and we postpone this sign problem for later analysis. We consider the integers Q_i and polynomials $H_{i,j}(x) \in \mathbb{Z}[x]$ defined as follows:

$$Q_i \stackrel{\text{def}}{=} \left(\frac{P}{p_i}\right)^\lambda = \prod_{\substack{p \in \mathcal{P}, \\ p \neq p_i}} p^\lambda,$$

$$H_{i,j} \stackrel{\text{def}}{=} \frac{f(x)}{(x - \tilde{r}_{i,j})} = \prod_{j' \neq j} (x - \tilde{r}_{i,j'}).$$

These polynomials are chosen so as to verify:

[3] The density of such primes is $1/\#\operatorname{Gal}(f)$, again by Čebotarev's density theorem.

Proposition 1 *Let* $T_{i,j} = T(\tilde{r}_{i,j}) \bmod p_i^\lambda$. *Then:*

$$T(x) = \left(\sum_{i,j} Q_i H_{i,j}(x) T_{i,j} \frac{1}{Q_i f'(\tilde{r}_{i,j})} \right) \bmod P^\lambda.$$

In the statement above, the inverse is to be understood in the p_i-adic ring $\mathbb{Z}_{p_i}$, or more precisely modulo p_i^λ. The proof is straightforward. Remark that $Q_i \bmod p_{i'}^\lambda$ cancels for $i' \neq i$, and that in the case $i = i'$, $H_{i,j}(\tilde{r}_{i',j'}) \bmod p_{i'}^\lambda$ is exactly equal to $f'(\tilde{r}_{i,j})$ for $j' = j$, and cancels otherwise. Then the bound $M \leq P^\lambda$ on the coefficients of T yields the announced result.

4.2 Determining signs

Proposition 1 allows to reconstruct $T(x)$ as soon as all $T_{i,j}$ are known. However, from the computation of roots modulo p_i, we only know $T'_{i,j} = s_{i,j} T_{i,j}$. Still, by examining the coefficient[4] of degree $d-1$ in $T(x)$, we obtain the following identity, which is derived from the expression in Proposition 1:

$$\begin{aligned}
[x^{d-1}]T(x) &= \sum_{i,j} Q_i T_{i,j} \frac{1}{Q_i f'(\tilde{r}_{i,j})} \bmod P^\lambda, \\
\frac{1}{P^\lambda}[x^{d-1}]T(x) &= \sum_{i,j} \frac{1}{p_i^\lambda} \left(T_{i,j} \frac{1}{Q_i f'(\tilde{r}_{i,j})} \bmod p_i^\lambda \right) \bmod 1.
\end{aligned}$$

Let now $x_{i,j}$ and $y_{i,j}$ be the real numbers

$$\begin{aligned}
x_{i,j} &= \frac{1}{p_i^\lambda} \left(T_{i,j} \frac{1}{Q_i f'(\tilde{r}_{i,j})} \bmod p_i^\lambda \right) \in [0,1[, \\
y_{i,j} &= \frac{1}{p_i^\lambda} \left(T'_{i,j} \frac{1}{Q_i f'(\tilde{r}_{i,j})} \bmod p_i^\lambda \right) \in [0,1[, \\
&\equiv \pm x_{i,j} \mod 1.
\end{aligned}$$

We can show that the sum of the $x_{i,j}$ numbers is exceptionally close to an integer. Indeed, the coefficient $[x^{d-1}]T(x)$ is at most M. Thus we have:

$$\left| \frac{1}{P^\lambda}[x^{d-1}]T(x) \right| \leq MP^{-\lambda} \leq \epsilon,$$

where ϵ is the arbitrary parameter introduced above. As a consequence, we have:

$$\sum_{i,j} x_{i,j} \in [-\epsilon, \epsilon] + \mathbb{Z}, \qquad \sum_{i,j} s_{i,j} y_{i,j} \in [-\epsilon, \epsilon] + \mathbb{Z}.$$

[4] Any coefficient of $T(x)$ may be considered. The only special thing about degree $d-1$ is that the corresponding expression is shorter to write.

This implies that by solving a knapsack-like problem, we may find which is the "right" combination of the $y_{i,j}$ coefficients, thereby solving the sign indetermination problem. Such problems are hard to solve, and are discussed further down. Let us only mention briefly that at least for modest numbers of primes, this problem remains practical.

Once the $s_{i,j}$ have been computed, we deduce the integer κ_{d-1} close to $\sum_{i,j} x_{i,j}$. We also obtain $T_{i,j} = s_{i,j}T'_{i,j}$. This gives:

$$\frac{1}{P^\lambda}[x^{d-1}]T(x) = \sum_{i,j} \frac{1}{p_i^\lambda} \left(s_{i,j}T'_{i,j} \frac{1}{Q_i f'(\tilde{r}_{i,j})} \bmod p_i^\lambda \right) - \kappa_{d-1}.$$

We can generalize this approach to now derive information relative to all coefficients of $T(x)$. This requires taking into account the interpolating polynomials $H_{i,j}(x)$. We define

$$\begin{aligned} c_{i,j,k} &= [x^k] \left(T'_{i,j} \frac{H_{i,j}(x)}{Q_i f'(\tilde{r}_{i,j})} \bmod p_i^\lambda \right), \\ c^*_{i,j,k} &= s_{i,j} c_{i,j,k}. \end{aligned}$$

The coefficients $c_{i,j,k}$ generalize the notations $x_{i,j}$ and $y_{i,j}$ above, since we have

$$x_{i,j} = \frac{c^*_{i,j,d-1}}{p_i^\lambda}, \quad \text{and} \quad y_{i,j} = \frac{c_{i,j,d-1}}{p_i^\lambda}.$$

We may use these notations to write a generalization of the expression above, and likewise for the expression of Proposition 1:

$$\begin{aligned} \frac{1}{P^\lambda}[x^k]T(x) &= \sum_{i,j} \frac{1}{p_i^\lambda}[x^k] s_{i,j} c_{i,j,k} - \kappa_k, \\ T(x) &= \sum_{i,j,k} x^k \left(Q_i s_{i,j} c_{i,j,k} - \kappa_k P^\lambda \right). \end{aligned}$$

Finally, we recall that the aim of the square root computation step is not really to compute $T(x)$, but instead its evaluation $T(m) \bmod N$, as already stated previously. As a consequence of the formula above, we are thus interested in $c_{i,j,k}m^k \bmod N$, which is a priori significantly smaller than p_i^λ.

4.3 Strategies for fast computation

The algorithm sketched so far needs some tuning, because the split into several computations, if done incorrectly, may in fact do more harm than good with respect to the overall complexity. The first important concern is the computation of the values $S(\tilde{r}_{i,j})$. Recall that these are $d\ell$ B-bit integers. In order to compute these values efficiently, one may proceed as follows. First compute $S(\alpha)$ with a subproduct tree (see e.g. [7, § 10.1]). Then proceed by computing reductions of all d coefficients modulo the ℓ prime powers p_i^λ. This multimodular reduction step

Algorithm crtalgsqrt$(N, m, f, \mathcal{S})$

INPUT: f monic irreducible, defining $K = \mathbb{Q}(\alpha)$,
N integer,
m a root of f modulo N.
$\mathcal{S}$ set of pairs (a, b) such that $S(\alpha) = \prod(a - b\alpha) \in \mathbb{Z}[\alpha]^2$.
PARAMETERS: $\ell = r \times t$, number of primes to consider.
OUTPUT: $T(m) \bmod N$, where $T(\alpha)^2 = S(\alpha)$.

1. Choose t sets $\mathcal{P}_1, \ldots, \mathcal{P}_t$ of r primes totally split in K.
   Partition $\mathcal{S}$ into t disjoints subsets $\mathcal{S}_1, \ldots, \mathcal{S}_t$.
2. For $k = 1, \ldots, t$, read $\mathcal{S}_k$. Deduce M and λ. Compute $S_k(x) = \prod_{(a,b)\in\mathcal{S}_k}(a - bx) \bmod f$.
3. For $i = 1, \ldots, \ell$ compute: p_i^λ, Q_i, $\frac{1}{Q_i} \bmod p_i^\lambda$, as well as for $j = 1, \ldots, d$, the root $r_{i,j} \bmod p_i$ of f, the lift $\tilde{r}_{i,j}$ with precision λ, as well as $\frac{H_{i,j}(x)}{f'(\tilde{r}_{i,j})} \bmod p_i^\lambda$.
4. For $\tau = 1, \ldots, t$:
   4.1. Compute $(S_\sigma(x) \bmod \mathcal{P}_\tau \stackrel{\text{def}}{=} \{S_\sigma(x) \bmod p,\ p \in \mathcal{P}_\tau\})_{\sigma=1\ldots t}$.
   4.2. For each $p_i \in \mathcal{P}_\tau$, and for $j = 1, \ldots, d$, compute:
      4.2.1 the evaluations $S_\sigma(\tilde{r}_{i,j}) \bmod p_i^\lambda$ for $\sigma = 1, \ldots, t$,
      4.2.2 the products $S(\tilde{r}_{i,j})$,
      4.2.3 the square roots $T'_{i,j} = \sqrt{S(\tilde{r}_{i,j})} \bmod p_i^\lambda$.
      4.2.4 the coefficients $c_{i,j,k} = [x^k]\left(T'_{i,j}\frac{H_{i,j}(x)}{Q_i f'(\tilde{r}_{i,j})} \bmod p_i^\lambda\right)$, as well as $c_{i,j,k}/p_i^\lambda \in \mathbb{R}$, and $c_{i,j,k}m^k \bmod N$.
5. Find the signs $s_{i,j}$ and the integer κ_{d-1} such that $\left|\sum_{i,j} s_{i,j}(c_{i,j,d-1}/p_i^\lambda) - \kappa_{d-1}\right| \leq \epsilon$. Deduce the integers $\kappa_0, \ldots, \kappa_{d-2}$.
6. return $\sum_{i,j,k} Q_i s_{i,j} c_{i,j,k} m^k - \kappa_k P^\lambda \bmod N$.

Algorithm 1: NFS square root using lifting and CRT.

may again be achieved with a subproduct tree. Finally, the evaluations modulo all $\tilde{r}_{i,j}$ is again a multi-evaluation, albeit relatively shallow, since we have only d evaluation points.

For an n-bit input, recall that we have set $\ell B \approx n/2$. Hence the first two steps above respectively have complexity $O(d^2\mathsf{M}(n)\log n)$ and $O(d\mathsf{M}(n)\log n)$. The multi-evaluation at all $\tilde{r}_{i,j}$ has complexity $O(\ell d^2\mathsf{M}(B))$.

This algorithm allows some trivial limited parallelism. We can achieve a t-fold reduction of the space complexity for the computation of the values $S(\tilde{r}_{i,j})$, and likewise for the time complexity, using t^2 nodes. This is done by splitting both $\mathcal{S}$ and $\mathcal{P}$ into t equally sized parts denoted by $\mathcal{S}_1, \ldots, \mathcal{S}_t$ and $\mathcal{P}_1, \ldots, \mathcal{P}_t$, respectively.

Corresponding to each set $\mathcal{S}_\sigma$, we define the polynomial $S_\sigma(x)$ as being the product

$$S_\sigma(x) = \prod_{(a,b)\in\mathcal{S}_\sigma} (a - bx) \bmod f.$$

For each pair of indices (σ, τ), we define the r-uple:

$$S_\sigma(x) \bmod \mathcal{P}_\tau \stackrel{\text{def}}{=} \{S_\sigma(x) \bmod p,\ p \in \mathcal{P}_\tau\}.$$

Each of t^2 nodes, indexed by (σ, τ), may compute the quantity above. The time and space complexity on each node are thus $(1/t)$-th of the total amount given above.

Once the values $S_\sigma(\tilde{r}_{i,j})$ have been computed, the values $S(\tilde{r}_{i,j})$ corresponding to the complete input set $\mathcal{S}$ can be obtained as the product over all σ.

The steps we have just detailed form the core of algorithm 1, namely steps 4.1, 4.2.1, and 4.2.2. Other steps of this algorithm are not detailed at length here. In algorithm 1, we have chosen to present the parallel version on up to t^2 nodes, but instantiating with $t = 1$ gives the version which is best suited for a sequential implementation.

4.4 Complexity

Table 1 gives complexity estimates for the different steps of the algorithm. Concerning parallelization, steps 3, 4.2.3, and 4.2.4 clearly scale to up to ℓ nodes easily. We have shown in the previous paragraphs how steps 4.1 to 4.2.2 may be improved t-fold in a quite simple way when run over t^2 nodes, neglecting communication costs.

The complexity of step 5 of algorithm 1 is related to a knapsack-like problem. It is straightforward to solve such a problem in time $2^{\frac{1}{2}d\ell}$, and the best known approach is $2^{0.313d\ell}$ [9].

In total, the algorithm proposed has a time and space complexity of the order of $O(d^2\mathsf{M}(n)\log n)$, which is similar to the lifting approach of Section 1 (when counting the computation of $S(\alpha)$ in the complexity).

Step	Time	Space
1	$O(\ell)$	$O(\ell)$
2	$O(d^2\mathsf{M}(n))$	$O(dn)$
3	$O(\ell d^2\mathsf{M}(B))$	$O(dn)$
4.1	$O(d\mathsf{M}(n)\log n)$	$O(dn\log n)$
4.2.1	$O(\ell d^2\mathsf{M}(B))$	$O(dn)$
4.2.2	$O(\ell d\mathsf{M}(B))$	$O(dn)$
4.2.3, 4.2.4	$O(\ell d\mathsf{M}(B))$	$O(dn)$
5	$O(2^{d\ell/2})$	$O(2^{d\ell/2})$
6	$O(\ell)$	$O(\ell)$

Table 1. Time and space complexity of the different steps of algorithm 1 (we have $rt = \ell$ and $rtB \approx n/2$).

4.5 Implementation and experimental data

Algorithm 1 has been implemented in `cado-nfs` [8]. For a practical application, the choice of parameters r and t is chiefly limited by the complexity of step 5 of the algorithm, since above some dimension, the knapsack problem becomes intractable. This algorithm has been run for the RSA-768 square root calculation, with the following parameters: $d = 6$, $t = 3$, $r = 2$, the total input size being 21 GB. On 18 nodes equipped with two 4-core Intel Xeon E5520 processors, and 32 GB of RAM, the computation claimed 6 hours. In comparison, using Montgomery's algorithm, the same computation on different hardware, but with the same number of 144 cores, took 4 hours, as it was reported in [11].

The difference in timings illustrates the difference in complexity. Montgomery's algorithm is linear, while the CRT-based algorithm we propose is quasi-linear. Because Montgomery's algorithm needs to exploit more accurate input data (namely, the ideal factorization of each ideal $(a - b\alpha)\mathcal{O}_K$), the time for reading this extended input set is significant. If we were to ignore I/O costs, the difference would be even more visible. Overall, the timings here are thus unsurprising. The CRT algorithm proposed here, compared to Montgomery's, has the advantage of not requiring the knowledge of the complete ideal factorization. In the perspective of a complete NFS implementation, this is interesting in that it simplifies the data flow, and also removes the need for an implementation of accurate computation of ideal valuations at *all* prime ideals. In the context of some NFS implementations which so far have chosen to restrict to the direct approach and avoid this step [8, 17], the CRT algorithm presented here offers a viable alternative.

4.6 A variant using a large number of primes

A variant of the algorithm above may be employed in order to handle a larger number of primes (for example up to 10 000). To this end, it is important to avoid the knapsack reconstruction step. An idea found in [6] allows to work around this issue in the case where inert primes can be found. Let $\mathcal{P}$ be the set of primes selected for the reconstruction. We assume each is chosen with exponent $\lambda = 1$, although varying this parameter is possible. As done previously, we denote by $T_p(x) = T(x) \bmod p$, for some $p \in \mathcal{P}$, and also $T'_p(x)$ the expression of the square root which is computed. Let s_p be the sign such that $T'_p(x) = s_p T(x) \bmod p$.

Let us focus on one particular coefficient of $T(x)$, for example that of degree $d-1$, which we denote by τ. Similarly to T_p and T'_p, we define the notations $\tau_p = \tau \bmod p$ and $\tau'_p = s_p\tau \bmod p$. Lacking the knowledge of s_p, we cannot distinguish between $\tau \bmod p$ and $-\tau \bmod p$. However, the *set* $\{\tau \bmod p, -\tau \bmod p\}$ is well determined by the computation of τ'_p. We may thus compute unambiguously $\tau^2 \bmod p = (\tau'_p)^2 \bmod p$. If this computation is done modulo many primes p, we can deduce the integer τ^2. To this end, we must have $\prod_{p\in\mathcal{P}} p \geq M^2$, where M is the bound on the coefficients of $T(x)$. The integer τ itself is then obtained by the square root of an n-bit integer, which is significantly cheaper than the

algebraic square root we are computing here. A byproduct of this computation is the set of signs $\{s_p\}$, which allows to complete the calculation.

This variant may also apply in a case without inert primes. In such a case, let $H \subset \mathfrak{S}_d$ be a representative set for the quotient $\mathrm{Gal}(L/\mathbb{Q})/\mathrm{Gal}(L/K)$, where L is the normal closure of K. Let $\sigma \in H$ be an element decomposing into a minimal number of distinct cycles, and let γ be this minimum. We want to use primes whose splitting pattern in K matches the cycle decomposition of σ. From Čebotarev's density theorem, it follows that the density of such primes is the density of this cycle decomposition pattern in H, which in particular is positive. If, as before, we try to relate $\tau \bmod p$ with the γ distinct coefficients obtained modulo each of the prime ideals above p, we have $\tau = \pm\tau_p'^{(1)} \pm \ldots \pm \tau_p'^{(\gamma)}$. There are 2^γ possible combinations. By evaluating the elementary symmetric functions on the 2^γ possible solutions modulo each p, we obtain τ as the root of an integer polynomial of degree 2^γ. The worst case for this extension is when γ is large: γ may reach $d/2$ in the case of Swinnerton-Dyer polynomials. We recover unsurprisingly the hard case of factoring polynomials over number fields.

References

[1] R. P. Brent, *Multiple-precision zero-finding methods and the complexity of elementary function evaluation*. In J. F. Traub (ed.), Analytic computational complexity, 151–176. Academic Press, New York, NY, 1975. Available at `http://web.comlab.ox.ac.uk/oucl/work/richard.brent/ftp/rpb028.ps.gz`. ↑ 3, 4

[2] R. Brent and P. Zimmermann, *Modern Computer Arithmetic*, Cambridge Monographs on Applied and Computational Mathematics, vol. 18, Cambridge University Press, 2010. ↑ 3, 4

[3] J. P. Buhler, A. K. Lenstra, and J. M. Pollard, *Factoring integers with the number field sieve*. In A. K. Lenstra and H. W. Lenstra Jr (eds.), The development of the number field sieve, vol. 1554 of *Lecture Notes in Math.*, 50–94. Springer–Verlag, 1993. ↑ 1

[4] H. Cohen, *A course in algorithmic algebraic number theory*, Grad. Texts in Math., vol. 138, Springer–Verlag, 1993. ↑ 2, 7

[5] J.-M. Couveignes, *Computing a square root for the number field sieve*. In A. K. Lenstra and H. W. Lenstra Jr (eds.), The development of the number field sieve, vol. 1554 of *Lecture Notes in Math.*, 95–102. Springer–Verlag, 1993. ↑ 1, 5

[6] A. Enge and A. V. Sutherland, *Class invariants by the CRT method*. In G. Hanrot, F. Morain, and E. Thomé (eds.), ANTS-IX, vol. 6197 of *Lecture Notes in Comput. Sci.*, 142–156. Springer–Verlag, 2010. Proceedings. ↑ 9, 15

[7] J. von zur Gathen and J. Gerhard, *Modern computer algebra*, Cambridge University Press, Cambridge, England, 1999. ↑ 12

[8] P. Gaudry, A. Kruppa, F. Morain, L. Muller, E. Thomé, and P. Zimmermann, `cado-nfs`, *An Implementation of the Number Field Sieve Algorithm*, 2011. Available at `http://cado-nfs.gforge.inria.fr/`. Release `1.1`. ↑ 2, 9, 15

[9] N. Howgrave-Graham and A. Joux, *New generic algorithms for hard knapsacks*. In Henri Gilbert (ed.), EUROCRYPT, vol. 6110 of *Lecture Notes in Comput. Sci.*, 235–256. Springer–Verlag, 2010. ↑ 14

[10] A. Joux, D. Naccache, and E. Thomé, *When e-th roots become easier than factoring*. In K. Kurosawa (ed.), ASIACRYPT 2007, vol. 4833 of *Lecture Notes in Comput. Sci.*, 13–28. Springer–Verlag, 2008. Proceedings. ↑ 2, 7, 9

[11] T. Kleinjung, K. Aoki, J. Franke, A. K. Lenstra, E. Thomé, J. Bos, P. Gaudry, A. Kruppa, P. L. Montgomery, D. A. Osvik, H. te Riele, A. Timofeev, and P. Zimmermann, *Factorization of a 768-bit RSA modulus*. In T. Rabin (ed.), CRYPTO 2010, vol. 6223 of *Lecture Notes in Comput. Sci.*, 333–350. Springer–Verlag, 2010. Proceedings. ↑ 2, 15

[12] H. W. Lenstra Jr and P. Stevenhagen, *Chebotarëv and his density theorem*, Math. Intelligencer **18**(2) (1996), 26–37. ↑ 5

[13] C. Monico, `ggnfs`, *A Number Field Sieve Implementation*, 2004–2005. Available at `http://www.math.ttu.edu/~cmonico/software/ggnfs/`. Release `0.77`. ↑ 2, 9

[14] P. L. Montgomery, *Square roots of products of algebraic numbers*. In W. Gautschi (ed.), Mathematics of Computation 1943–1993 : a Half-Century of Computational Mathematics, vol. 48 of *Proc. Sympos. Appl. Math.*, 567–571. Amer. Math. Soc., 1994. ↑ 1, 6, 7, 17

[15] P. L. Montgomery, *Square roots of products of algebraic numbers*, 1997. Unpublished draft, significantly different from published version [14]. Dated May 16, 1997. ↑ 1, 6, 7, 9

[16] P. Q. Nguyen, *A Montgomery-like square root for the number field sieve*. In J. P. Buhler (ed.), ANTS-III, vol. 1423 of *Lecture Notes in Comput. Sci.*, 151–168. Springer–Verlag, 1998. Proceedings. ↑ 7

[17] J. Papadopoulos, `msieve`, *A Library for Factoring Large Integers – `release 1.50`*, 2004–. Available at `http://www.boo.net/~jasonp`. Release `1.50`. ↑ 2, 15

[18] A. V. Sutherland, *Accelerating the CM method*, 2012. Available at `http://arxiv.org/abs/1009.1082`. Preprint. ↑ 9