Deterministic root finding over finite fields using Graeffe transforms

Bruno Grenet[A]

Laboratoire d'informatique, de robotique
et de microélectronique de Montpellier
LIRMM, UMR 5506 CNRS, CC477
Université Montpellier 2
161, rue Ada
34095 Montpellier Cedex 5, France

Email: bruno.grenet@lirmm.fr

Joris van der Hoeven[b], Grégoire Lecerf[c]

Laboratoire d'informatique de l'École polytechnique
LIX, UMR 7161 CNRS
Campus de l'École polytechnique
1, rue Honoré d'Estienne d'Orves
Bâtiment Alan Turing, CS35003
91120 Palaiseau, France

b. Email: vdhoeven@lix.polytechnique.fr
c. Email: lecerf@lix.polytechnique.fr

Version of January 13, 2015

We design new deterministic algorithms, based on Graeffe transforms, to compute all the roots of a polynomial which splits over a finite field $\mathbb{F}_q$. Our algorithms were designed to be particularly efficient in the case when the cardinality $q-1$ of the multiplicative group of $\mathbb{F}_q$ is smooth. Such fields are often used in practice because they support fast discrete Fourier transforms. We also present a new nearly optimal algorithm for computing characteristic polynomials of multiplication endomorphisms in finite field extensions. This algorithm allows for the efficient computation of Graeffe transforms of arbitrary orders.

1. Introduction

Let $\mathbb{F}_q$ represent the finite field with $q=p^k$ elements, where p is a prime number, and $k\geqslant 1$. Throughout this article, such a field is supposed to be described as a quotient of $\mathbb{F}_p[x]$ by a *monic* irreducible polynomial. Let $f\in\mathbb{F}_q[x]$ represent a *separable monic polynomial* of degree $d\geqslant 1$ which *splits* over $\mathbb{F}_q$, which means that all its irreducible factors have degree one and multiplicity one. In this article we are interested in computing all the roots of f.

1.1. Motivation

In a previous paper [20], we presented efficient randomized root finding algorithms. Such algorithms were needed for the efficient interpolation, into the standard monomial basis, of polynomials that are given through evaluation functions. This latter problem is also known

A. Partially supported by a LIX–Qualcomm®–Carnot postdoctoral fellowship.

under the name *sparse interpolation*, and root finding often turns out to be a bottleneck, as indicated in [23]. In fact, in this case, the ground field can be chosen to be $\mathbb{F}_p$ with $p = M\,2^m + 1$, and where 2^m is taken to be much larger than the number of terms to be discovered. In order to minimize the size of p, making it fit into a machine register, we usually take $M = O(\log p)$ as small as possible. A typical example is $p = 7 \cdot 2^{26} + 1$. We informally refer to such primes as *FFT primes*.

While working on [20], it turned out that some of the new methods could also be used for the design of fast deterministic methods for computing all the roots of f. Even though randomized algorithms are more useful in practice, the existence of efficient deterministic algorithms remains interesting from a theoretical point of view. The goal of this paper is to report on our progress on this issue.

1.2. Notations and prerequisites

We will use the same notations as in [20], which we briefly recall now. The *multiplicative group* $\mathbb{F}_q \setminus \{0\}$ of $\mathbb{F}_q$ is written $\mathbb{F}_q^*$. Complexity bounds will frequently be expressed using the *soft-Oh* notation: $f(n) \in \tilde{O}(g(n))$ means that $f(n) = g(n) \log^{O(1)} g(n)$. Given $x \in \mathbb{R}$, we write $\lfloor x \rfloor = \max\{k \in \mathbb{Z} : k \leqslant x\}$ and $\lceil x \rceil = \min\{k \in \mathbb{Z} : k \geqslant x\}$. The *remainder* of a division of g by f is denoted by $g \operatorname{rem} f$.

We write $\mathsf{M}(d)$ for the complexity of multiplication of polynomials of degree $<d$ over an arbitrary ring. We also write $\mathsf{I}(n)$ and $\mathsf{M}_q(d)$ for the bit complexities (in the Turing machine model) of n-bit integer multiplication and multiplication of polynomials of degrees $<d$ over $\mathbb{F}_q$. We make the customary assumption that $\mathsf{M}(d)/d$, $\mathsf{I}(n)/n$ and $\mathsf{M}_q(d)/d$ are increasing functions. Currently best known bounds [10, 21, 22] are $\mathsf{M}(d) = O(d \log d \log\log d)$, $\mathsf{I}(n) = O(n \log n\, 8^{\log^* n})$ and $\mathsf{M}_q(d) = O\big(d \log q \log(d \log q)\, 8^{\log^*(d \log q)}\big)$, where $\log^*$ stands for the *iterated logarithm* function $\log^* n = \min\big\{k \in \mathbb{N} : \log^{k\times} \log x \leqslant 1\big\}$.

We freely use the following classical facts: ring operations in $\mathbb{F}_p$ cost $O(\mathsf{I}(\log p))$ and one division or inversion in $\mathbb{F}_p$ costs $O(\mathsf{I}(\log p) \log\log p)$ [17, Chapter 11]. The ring operations in $\mathbb{F}_q$ cost at most $O(\mathsf{M}_p(k))$, and inversions take at most $O(\mathsf{M}_p(k) \log k + \mathsf{I}(\log p) \log\log p)$. For convenience, m_q and d_q will respectively denote cost functions for the product and the inverse in $\mathbb{F}_q$. The gcd of two polynomials of degrees at most d over $\mathbb{F}_q$ can be computed in time $O(\mathsf{M}_q(d) \log d)$ [17, Algorithm 11.4]. Given polynomials f and $g_1, \ldots, g_l$ monic with $\deg f = d$ and $\deg g_1 + \cdots + \deg g_l = O(d)$, all the remainders $f \operatorname{rem} g_i$ can be computed in time $O(\mathsf{M}_q(d) \log l)$ [17, Chapter 10]. The inverse problem, called *Chinese remaindering*, can be solved within a similar cost $O(\mathsf{M}_q(d) \log l + d\,\mathsf{d}_q)$.

For an integer n, the largest prime dividing n is written $S_1(n)$, and the second one $S_2(n)$. Pollard and Strassen have given a deterministic algorithm for factoring $p-1$, of bit complexity $O\big(\mathsf{M}\big(S_2(p-1)^{1/2}\big)\,\mathsf{I}(\log p) \log p\big)$ (see for instance [17, Corollary 19.4]). Ignoring smoothness considerations, the latter result has been improved into $O\big(\mathsf{I}\big(p^{1/4} \log p\big)\big)$ in [6], and further refined to $O\big(\mathsf{I}\big(p^{1/4} (\log p)/\sqrt{\log\log p}\big)\big)$ in [13].

In this article, when needed, we consider that the factorization of $q-1$ and a primitive element of $\mathbb{F}_q^*$ have been precomputed once, and we discard the necessary underlying costs. In practice, if the factorization of $q-1$ is given, then it is straightforward to verify whether a given element is primitive. For known complexity bounds and historical details on these tasks, we refer to [2, 33, 44].

1.3. Related work

It is now classical that polynomial factorization over $\mathbb{F}_q$ reduces to finding roots over $\mathbb{F}_p$ in deterministic polynomial time, uniformly in d and $\log q$. But it is still unknown whether root finding can be solved in deterministic polynomial time or not, even under classical

conjectures in number theory. This problem can nevertheless be solved efficiently by randomized algorithms in average polynomial time. Concerning related results and historical surveys on this topic, the reader might consult [17, 18, 25, 33].

Seminal algorithms for polynomial factorization over finite fields are classically attributed to Berlekamp [3, 4], and Cantor and Zassenhaus [11], but central earlier ideas can be found in works of Gauss, Galois, Arwins, Faddeev and Skopin. Cantor–Zassenhaus' algorithm is randomized and well suited to compute roots of polynomials of degree d that split over $\mathbb{F}_q$ in average time $O(\mathsf{M}_q(d)\,(\log q + \log d)\log d + d\,\mathsf{d}_q)$. Of course, if $q = O(d)$ then an exhaustive search can be naively performed in time $O(\mathsf{M}_q(d)\log d)$ (the factor $\log d$ can be discarded if a primitive element of $\mathbb{F}_q^*$ is given, by means of [8, Proposition 3]), so that the cost of root finding simplifies to $O(\mathsf{M}_q(d)\log q \log d + d\,\mathsf{d}_q)$. This classical approach is for instance implemented in the NTL library written by Shoup [45]. However neither Berlekamp's nor Cantor–Zassenhaus' algorithm seems to benefit of particular prime numbers such as FFT primes. Instead, alternative approaches have been proposed by Moenck [32], von zur Gathen [16], Mignotte and Schnorr [31], and then by Rónyai [37]. Recent slight improvements of Moenck's and Mignotte–Schnorr's algorithms may also be found in [20].

Deterministic polynomial time root finding has been tackled by several authors. Camion showed that, for a fixed finite field, one can pre-compute a suitable subset which allows to factor any polynomial in deterministic polynomial time [9]. Nevertheless the construction of such a subset is purely theoretical, and it is not clear that there exists an efficient algorithm to compute it, even for FFT prime fields.

Schoof, Rónyai, Huang, and Źrałek designed different methods for particular types of input polynomials according to their syntax or to properties of the Galois group of the lifted input polynomial over $\mathbb{Q}$ [24, 37, 38, 40, 46]. Some of their algorithms require the general Riemann hypothesis (GRH) or the extended Riemann hypothesis (ERH). Sub-exponential algorithms are also known from the work of Evdokimov [14], which has recently led to new conjectures in terms of graphs [1]. Other conjectural algorithms can be found in [15, 39].

Another series of articles concern complexity bounds which are uniformly polynomial in the degree and in $S_1(q-1)$. Such algorithms are practical when $q-1 = p_1^{m_1}\cdots p_r^{m_r}$ is sufficiently smooth. Assuming a primitive element of $\mathbb{F}_q^*$ is given, Mignotte and Schnorr [31] proposed a method based on a cascade of gcds, that needs $O(\mathsf{M}(d)\sum_{i=1}^{r} m_i\,(\log q + p_i \log d))$ operations in $\mathbb{F}_q$. The computation of a primitive element might of course be expensive but it can be seen as a pre-computation. In fact, von zur Gathen proved the deterministic polynomial time equivalence (in terms of $S_1(p-1)$ and input size) between the computation of primitive elements and polynomial factorization [16].

Rónyai [37] obtained a polynomial complexity bound in $S_1(p-1)$, d and $\log p$ by means of linear algebra techniques, using primitive roots of orders lower than in [16], but he did not explicit the exponents in the bound. For $\mathbb{F}_p[x]$, Shoup [41] reached the bound $p^{1/2}\log^2 p\,\tilde{O}(d^2)$ in terms of the number of operations in $\mathbb{F}_p$, and then refined it to $p^{1/2}\log p\,\tilde{O}(d\,k\min(d,k)) + \tilde{O}(d\,k^2)\log p$ for $\mathbb{F}_q[x]$, still in terms of operations in $\mathbb{F}_p$ [42]. Finally Shoup proved the bound $S_1(p-1)^{1/2}\,(d\log p)^{O(1)}$ in $\mathbb{F}_p[x]$ under ERH [43].

1.4. Our contributions

The main contribution of this article is an efficient deterministic root finding method. The new algorithm is based on generalized Graeffe transforms and it is particularly efficient for finite fields $\mathbb{F}_p$ such that p is an FFT prime. Roughly speaking, the generalized Graeffe transform replaces modular exponention as used in Cantor–Zassenhaus' algorithm, and gcd computations are changed to multi-point evaluations.

Thanks to a slight improvement of Shoup's adaptation of Pollard–Strassen's algorithm to find roots deterministically [43], we obtain a new deterministic complexity bound in terms of $S_1(p-1)^{1/2}$. In addition, in the smooth case $S_1(q-1)=O(\log q)$, the cost of our deterministic algorithm is softly equivalent to the one of Cantor–Zassenhaus.

The complexity of generalized Graeffe transforms is studied in Section 2. A first ingredient is the transposed modular composition algorithm of Kedlaya and Umans [27], also called modular power projections algorithm. The second ingredient is a special variant for finite fields of Newton–Girard's formulas for recovering a polynomial from the power sums of its roots. Such a variant had been previously designed by Bostan, Gonzalez-Vega, Perdry and Schost [7] in the case of prime finite fields. We extend their results to finite fields from scratch, using very simple arguments, and independently of the framework designed in [12].

2. Generalized Graeffe transforms over finite fields

Classically, the *Graeffe transform* of a polynomial $g \in \mathbb{F}_q[x]$ of degree d is the unique polynomial $h \in \mathbb{F}_q[x]$ satisfying $h(x^2) = g(x)\, g(-x)$. If $g(x) = \prod_{i=1}^{d} (\alpha_i - x)$, then $h(x) = \prod_{i=1}^{d} (\alpha_i^2 - x)$. This construction can be extended to higher orders as follows: the *generalized Graeffe transform* of g of order π, written $G_\pi(g)$, is defined as the resultant

$$G_\pi(g)(x) = (-1)^{\pi d} \mathrm{Res}_z(g(z), z^\pi - x).$$

If $g = \prod_{i=1}^{d} (\alpha_i - x)$, then $G_\pi(g)(x) = \prod_{i=1}^{d} (\alpha_i^\pi - x)$. Equivalently, $G_\pi(g)$ is the characteristic polynomial of multiplication by x^π in $\mathbb{F}_q[x]/(g)$ (up to the sign). In this section we show how to compute Graeffe transforms efficiently. We start with a particularly simple and efficient algorithm for the smooth case, repeated from [20].

2.1. Smooth case

For our root finding algorithm, the most important case is when $q-1$ is smooth and the order π of the generalized Graeffe transform divides $q-1$.

PROPOSITION 1. *Let $\pi_1, \ldots, \pi_m$ be integers $\geqslant 2$, such that $\chi = \pi_1 \cdots \pi_m$ divides $q-1$, and let ξ_i be given primitive roots of unity of order π_i, for all $i \in \{1, \ldots, m\}$. If g is a monic polynomial in $\mathbb{F}_q[x]$ of degree d, then the generalized Graeffe transforms of orders π_1, $\pi_1\,\pi_2, \pi_1\,\pi_2\,\pi_3, \ldots, \chi$ of g can be computed in time $O(m\,\mathsf{M}_q(\mu\,d))$ or $\sigma\,\tilde{O}(d \log q \log \mu)$, where $\mu = \max(\pi_1, \ldots, \pi_m)$ and $\sigma = \pi_1 + \cdots + \pi_m$.*

Proof. Writing $g(x) = c \prod_{j=1}^{d} (\alpha_j - x)$ in an algebraic closure of $\mathbb{F}_q$, the Graeffe transform of g of order π_i is $h_i(x) = c^{\pi_i} \prod_{j=1}^{d} (\alpha_j^{\pi_i} - x)$. Consequently this leads to $h_i(x^{\pi_i}) = g(x)\, g(\xi_i\, x)\, g(\xi_i^2\, x) \cdots g\big(\xi_i^{\pi_i - 1}\, x\big)$. Using the latter formula, by Lemma 2 below, the transform can be obtained in time $O(\mathsf{M}_q(\pi_i\, d))$. Taking the sum over i concludes the proof. □

LEMMA 2. *Let g be a polynomial of degree $d \geqslant 1$ in $\mathbb{F}_q[x]$, let $\alpha \in \mathbb{F}_q$, and let l be an integer. Then the product $P_l(x) = g(x)\, g(\alpha\, x)\, g(\alpha^2\, x) \cdots g(\alpha^{l-1}\, x)$ can be computed in time $O(\mathsf{M}_q(l\, d))$.*

Proof. Let $h := \lfloor l/2 \rfloor$. If l is even, then $P_l(x) = P_h(x)\ P_h(\alpha^h\ x)$, otherwise we have $P_l(x) = P_h(x)\ P_h(\alpha^h\ x)\ g(\alpha^{l-1}\ x)$. These formulas lead to an algorithm with the claimed cost. □

Notice that this lemma generalizes to arbitrary fields.

2.2. General case

In the general case, let $g \in \mathbb{F}_q[x]$ be monic of degree d. Given a polynomial e of degree $<d$, we are interested in computing the characteristic polynomial h of the multiplication by e in $R := \mathbb{F}_q[x]/(g)$. Our strategy essentially follows Le Verrier's method: we first compute the traces $\mathrm{Tr}(e^i)$ of the multiplication by e^i in R for all $i \in \{1, ..., d\}$, which are also called the *power sums* of the roots of the characteristic polynomial h. The generating series $\tau(x) = \sum_{i \geqslant 0} \mathrm{Tr}(e^i)\ x^i \in \mathbb{F}_q[[x]]$ of these power sums relates to the logarithmic derivative of the *reverse polynomial* $\eta = x^d\, h(1/x)$ of h *via* a first order differential equation

$$\frac{\eta'(x)}{\eta(x)} = \tau(x) + O(x^d). \tag{1}$$

This equation can be seen as a compact form of the classical Newton–Girard identities which relate power sums and symmetric polynomials. After computing τ it thus suffices to solve this equation in order to deduce h. If $p \geqslant d$, then this method is classical. If $p < d$, then h cannot directly be retrieved in this way [35, Fact 2.1]. In the rest of this section, we will overcome this problem by computing with p-adic integers at a suitable finite precision κ.

2.3. Power sums

Let $\mathbb{Z}/p^\kappa\,\mathbb{Z}$ be the ring of truncated p-adic integers at precision κ. It is convenient to write $\mathrm{GR}(p^\kappa, k)$ for the *Galois ring* of characteristic p^κ and degree k, which is classically defined as $\mathrm{GR}(p^\kappa, k) = ((\mathbb{Z}/p^\kappa\,\mathbb{Z})[z])/(\mu)$, where $\mu \in (\mathbb{Z}/p^\kappa\,\mathbb{Z})[z]$ is a monic polynomial of degree k and is irreducible modulo p.

Let g represent a given monic polynomial of degree d, let $A = \mathrm{GR}(p^\kappa, k)[x]/(g)$. We write $\mathrm{Tr}\colon A \to \mathrm{GR}(p^\kappa, k)$ the classical trace function on A. In order to compute the power sums $\mathrm{Tr}(e)$, $\mathrm{Tr}(e^2)$, ..., $\mathrm{Tr}(e^d)$, we will use the following proposition, which is essentially a consequence of a result due to Kedlaya and Umans in [26, 27]:

Proposition 3. *Let $\delta > 0$ be a fixed constant. If $d^{1+\delta} \leqslant q - 1$, then the traces $\mathrm{Tr}(e)$, $\mathrm{Tr}(e^2), ..., \mathrm{Tr}(e^d)$ for a given $e \in A$ can be computed in time $(d \log(q^\kappa))^{1+\delta}$.*

Proof. Let $\gamma(x) = x^d\, g(1/x)$ denote the reverse polynomial of g. Since g is monic we have $\gamma(0) = 1$. In the canonical basis made of the classes of $1, x, ..., x^{d-1}$ in A, the trace function can be computed as the first d terms of the generating series

$$\frac{\gamma'(x)}{\gamma(x)} = \mathrm{Tr}(x) + \mathrm{Tr}(x^2)\, x + \mathrm{Tr}(x^3)\, x^2 + \cdots + \mathrm{Tr}(x^d)\, x^{d-1} + O(x^d),$$

in softly linear time, namely $\tilde{O}(\kappa\, d \log q)$.

The cardinality of A is q^κ, and A contains at least $q-1$ invertible elements. Then, for every constant $\delta > 0$ such that $d^{1+\delta} \leqslant q - 1$, Theorem 7.7 of [27, p. 1792] provides us with an algorithm to compute $\mathrm{Tr}(e), \mathrm{Tr}(e^2), ..., \mathrm{Tr}(e^d)$ in time $d^{1+\delta} \log^{1+o(1)}(q^\kappa)$. □

Remark 4. For d close to q the proposition does not apply. A simple solution is to lift the computations in $\mathbb{F}_{q^2}$. Nevertheless, for finding the roots of g, this case is in fact favorable: it suffices to evaluate g at all the elements of $\mathbb{F}_q$. Therefore we will not need to work in $\mathbb{F}_{q^2}$.

2.4. Newton–Girard's identities

We carry on with the notations of the previous subsection. Let e be a given element in A, and let $\tau(x) = \sum_{i \geqslant 0} \mathrm{Tr}(e^i)\, x^i \in \mathrm{GR}(p^\kappa, k)[[x]]$ represent the generating series of the power sums of the roots of the characteristic polynomial h of e. If $p \geqslant d$, then h can be deduced from $\mathrm{Tr}(e), \mathrm{Tr}(e^2), ..., \mathrm{Tr}(e^d)$ in softly linear time, using Newton–Girard's identities (1), as for instance in [5, Corollary 1]. If $k = 1$, then Theorem 1 of [7] solves the equation modulo p whenever $\kappa \geqslant \lfloor \log d / \log p \rfloor + 1$. This subsection is dedicated to the remaining case when $p < d$ and k is general. Our new alternative proofs are also shorter than in [7].

LEMMA 5. *Let $\theta \in \mathrm{GR}(p^\kappa, k)[x]$ be of degree $\leqslant d$ satisfying $\theta(0) = 1$, and $\theta'(x)/\theta(x) = \tau(x) + O(x^d)$. If $\kappa > \lfloor \log d / \log p \rfloor$, then θ coincides with η modulo p at precision $O(x^{d+1})$.*

Proof. Let $\psi = \theta / \eta$. Writing ψ as $\sum_{i \geqslant 0} \psi_i\, x^i$, we deduce from $\psi' = O(x^d)$ that $i\, \psi_i = 0$ for all $i \in \{1, ..., d\}$, whence $p^{\kappa - \mathrm{val}_p\, i}$ divides ψ_i, where $\mathrm{val}_p\, i$ denotes the p-adic valuation of i. Since $\kappa > \lfloor \log d / \log p \rfloor$ is equivalent to $d < p^\kappa$, we have $\kappa - \mathrm{val}_p\, i > 0$, which concludes the proof. □

Denote cost of multiplication in $\mathrm{GR}(p^\kappa, k)[x]$ of polynomials of degrees $< d$ by $\mathsf{M}_{p^\kappa,k}(d)$. We have $\mathsf{M}_{p^\kappa,k}(d) = O\big(k\, \kappa\, d \log p \log(k\, \kappa\, d \log p)\, 8^{\log^*(k \kappa d \log p)}\big) = \tilde{O}(\kappa\, d \log q)$ according to [22].

PROPOSITION 6. *Let $e \in A$, and let $\kappa = \lfloor \log d / \log p \rfloor + 1$. From $\mathrm{Tr}(e^1), ..., \mathrm{Tr}(e^d)$, we can compute the characteristic polynomial h of e modulo p in time $O(\mathsf{M}_{p^\kappa,k}(d)) = \tilde{O}(d \log q)$.*

Proof. We will show how to compute a series $\theta \in \mathrm{GR}(p^\kappa, k)[[x]]$ satisfying $\theta(0) = 1$ and $\theta'(x)/\theta(x) = \tau(x) + O(x^d)$, using Newton's method. In view of the above lemma, this will yield η followed by h modulo p.

Suppose that we are given θ such that $\theta(0) = 1$ and $\theta'(x)/\theta(x) = \tau(x) + O(x^{n-1})$ for some integer $n \geqslant 1$. Then we claim that there exists $\tilde{\theta} \in \mathrm{GR}(p^\kappa, k)[[x]]$ of valuation in x at least n and such that

$$\tilde{\theta}'(x) = \tau(x) - \frac{\theta'(x)}{\theta(x)} + O(x^{2n-1}). \tag{2}$$

Indeed, using that $\tau - \theta'/\theta = O(x^{n-1})$ and $\tilde{\theta} = O(x^n)$, this equation is equivalent to

$$\frac{\tilde{\theta}'}{1 + \tilde{\theta}} = \left(\tau - \frac{\theta'}{\theta}\right)\left(1 - \tilde{\theta} + \tilde{\theta}^2 + \cdots\right) + O(x^{2n-1}) = \tau - \frac{\theta'}{\theta} + O(x^{2n-1}),$$

and therefore equivalent to

$$\frac{\big(\big(1 + \tilde{\theta}\big)\, \theta\big)'}{\big(1 + \tilde{\theta}\big)\, \theta} = \frac{\tilde{\theta}'}{1 + \tilde{\theta}} + \frac{\theta'}{\theta} = \tau(x) + O(x^{2n-1}).$$

This latter equation admits $\eta / \theta - 1$ as a solution. Having proved our claim, we can compute $\tilde{\theta}$ as being any integral of $\tau - \theta'/\theta$ modulo $O(x^{2n})$, and observe that $\hat{\theta} = \big(1 + \tilde{\theta}\big)\, \theta$ satisfies $\hat{\theta}(0) = 1$ and $\hat{\theta}'/\hat{\theta} = \tau + O(x^{2n-1})$.

The cost of one iteration (2) is bounded by $O(\mathsf{M}_{p^\kappa,k}(n))$. Starting with $n = 1$ and $\theta = 1$, we recursively apply the iteration for $n = 1, 2, 4, ..., 2^{\lceil \log \kappa / \log 2 \rceil - 1}$, the end result being a $\theta \in \mathrm{GR}(p^\kappa, k)[[x]]$ with $\theta'/\theta = \tau + O(x^d)$. It is classical that the cost of the last iteration dominates the total cost, which is therefore bounded by $O(\mathsf{M}_{p^\kappa,k}(d))$. □

2.5. Characteristic polynomials

Now we come back to the original computational problem over $\mathbb{F}_q$.

Theorem 7. *Let $g \in \mathbb{F}_q[x]$ be a monic polynomial of degree d, and let $e \in \mathbb{F}_q[x]$ be of degree $<d$. For every constant $\delta > 0$ such that $d^{1+\delta} \leqslant q-1$, the characteristic polynomial h of the multiplication by e in $\mathbb{F}_q[x]/(g)$ can be computed in time $(d \log q)^{1+\delta}$.*

Proof. We embed $\mathbb{F}_q[x]/(g)$ into $A = \mathrm{GR}(p^\kappa, k)[x]/(g)$, where $\mathrm{GR}(p^\kappa, k)$ is constructed from $(\mathbb{Z}/p^\kappa\mathbb{Z})[z]/(\mu)$, and $\mu \in (\mathbb{Z}/p^\kappa\mathbb{Z})[z]$ is the monic defining polynomial of degree k of $\mathbb{F}_q$ over $\mathbb{F}_p$.

Let $\delta > 0$ be such that $d^{1+\delta} \leqslant q-1$. Proposition 3 allows us to compute the traces of the successive powers of e seen in A in time $(d \log(q^\kappa))^{1+\delta}$. Then Proposition 6 allows to deduce h modulo p in softly linear time. □

Remark 8. If the characteristic p is larger than the multiplicities of the roots of h, then Pan's algorithm [35] does not involve p-adic arithmetic, but may require higher order traces.

Corollary 9. *Let g be a monic polynomial in $\mathbb{F}_q[x]$ of degree d. For every constant $\delta > 0$ such that $d^{1+\delta} \leqslant q-1$, the generalized Graeffe transform $G_\pi(g)$ can be computed in time $(d \log q)^{1+\delta} + \tilde{O}(d \log q \log \pi)$.*

Proof. We apply the latter theorem to $e(x) = x^\pi \operatorname{rem} g(x)$, that can be computed in time $O(\mathsf{M}_q(d) \log \pi)$. □

2.6. Tangent Graeffe transforms

Whenever a Graeffe transform preserves the number of distinct simple roots, the so called tangent Graeffe transform can be used to directly recover the original simple roots from the simple roots of the transformed polynomial. Our randomized root finding algorithms from [20] heavily rely on this fact. Unfortunately, we have not found a way to exploit tangent Graeffe transforms in the deterministic setting. For completeness, we will nevertheless show that the results of this section extend to the tangent case.

Introducing a formal parameter ε with $\varepsilon^2 = 0$, we define the *generalized tangent Graeffe transform* of g of order π as being $G_\pi(g(x+\varepsilon)) \in (\mathbb{F}_q[\varepsilon]/(\varepsilon^2))[x]$. For any ring R, computations with "tangent numbers" in $R[\varepsilon]/(\varepsilon^2)$ can be done with constant overhead with respect to computations in R (in the FFT model, the overhead is asymptotically limited to a factor of two).

In the context of the Graeffe method, the tangent transform is classical (for instance, see [30, 34] for history, references, and use in numerical algorithms). The generalized tangent Graeffe transform can also be seen as the *tangent characteristic polynomial* of x^π modulo $g(x+\varepsilon)$, and this construction is attributed to Kronecker in algebraic geometry [19, 28].

Proposition 1 from Section 2.1 admits a straightforward generalization to tangent Graeffe transforms with a constant overhead. In order to generalize Proposition 3, we introduce $A_\varepsilon = \mathrm{GR}(p^\kappa, k)[[\varepsilon]][x]/(g(x+\varepsilon))$, where $\mathrm{GR}(p^\kappa, k) = ((\mathbb{Z}/p^\kappa\mathbb{Z})[z])/(\mu)$, where $\mu \in (\mathbb{Z}/p^\kappa\mathbb{Z})[z]$ is monic of degree k and irreducible modulo p. Let Tr_ε represent the trace function in A_ε.

Proposition 10. *Let $\delta > 0$ be a fixed constant. If $d^{1+\delta} \leqslant q-1$, then the traces $\mathrm{Tr}_\varepsilon(e + \varepsilon\,\bar{e})$, $\mathrm{Tr}_\varepsilon((e+\varepsilon\,\bar{e})^2), \ldots, \mathrm{Tr}_\varepsilon((e+\varepsilon\,\bar{e})^d)$ modulo ε^2, for a given $e + \varepsilon\,\bar{e} \in A_\varepsilon$, can be computed in time $(d \log(q^\kappa))^{1+\delta}$.*

Proof. The value $\mathrm{Tr}_\varepsilon((e+\varepsilon\,\bar{e})^i)$ equals the trace of $(e(x-\varepsilon)+\varepsilon\,\bar{e}(x-\varepsilon))^i$ computed in $\mathrm{GR}(p^\kappa,k)[[\varepsilon]][x]/(g)$. Therefore we obtain:

$$\mathrm{Tr}_\varepsilon((e(x)+\varepsilon\,\bar{e}(x))^i) = \mathrm{Tr}((e(x-\varepsilon)+\varepsilon\,\bar{e}(x-\varepsilon))^i) = \mathrm{Tr}(e^i(x)+i\,\varepsilon\,(\bar{e}(x)-e'(x))\,e^{i-1}(x)).$$

We therefore use Theorem 7.7 of [27, p. 1792] twice over $A = \mathrm{GR}(p^\kappa,\, k)[x]\,/\,(g)$: once with Tr, and once for $\mathrm{Tr}\circ\varphi$, where φ is the product by $\bar{e}-e'$. □

THEOREM 11. *Let $g \in \mathbb{F}_q[x]$ be a monic polynomial of degree d, and let $e,\ \bar{e} \in \mathbb{F}_q[x]$ be of degrees $<d$. For every constant $\delta > 0$ such that $d^{1+\delta} \leqslant q-1$, the characteristic polynomial $h(x)+\varepsilon\,\bar{h}(x)$ of the multiplication by $e(x)+\varepsilon\,\bar{e}(x)$ in $\mathbb{F}_q[[\varepsilon]][x]/(g(x+\varepsilon))$ can be computed in time $(d\log q)^{1+\delta}$.*

Proof. We embed $\mathbb{F}_q[[\varepsilon]][x]/(g)$ into $A_\varepsilon = \mathrm{GR}(p^\kappa,k)[[\varepsilon]][x]/(g(x+\varepsilon))$, where $\mathrm{GR}(p^\kappa,k)$ is constructed from $(\mathbb{Z}/p^\kappa\,\mathbb{Z})[z]/(\mu)$, and $\mu \in (\mathbb{Z}/p^\kappa\,\mathbb{Z})[z]$ is the monic defining polynomial of degree k of $\mathbb{F}_q$ over $\mathbb{F}_p$.

Let $\delta > 0$ be such that $d^{1+\delta} \leqslant q-1$. Proposition 10 allows us to compute the traces of the successive powers of $e+\varepsilon\,\bar{e}$ seen in A_ε in time $(d\log(q^\kappa))^{1+\delta}$. Then Proposition 6 allows us to deduce $h+\varepsilon\,\bar{h}$ modulo p in softly linear time, *mutatis mutandis*. □

COROLLARY 12. *Let g be a monic polynomial in $\mathbb{F}_q[x]$ of degree d. For every constant $\delta > 0$ such that $d^{1+\delta} \leqslant q-1$, the generalized tangent Graeffe transform $G_\pi(g(x+\varepsilon))$ can be computed in time $(d\log q)^{1+\delta} + \tilde{O}(d\log q\log\pi)$.*

Proof. We apply the latter theorem to $e(x)+\varepsilon\,\bar{e}(x) = x^\pi \text{ rem } g(x+\varepsilon)$, that can be computed in time $O(\mathsf{M}_q(d)\log\pi)$. □

3. Root finding based on Graeffe transforms

Our new root finding algorithm is technically reminiscent of numerical solvers: we make use of the Graeffe transform instead of the modular exponentiation, and replace gcds by multi-point evaluations. We begin this section with a first simple version of the algorithm, and then introduce new ingredients in order to improve it. Throughout this section, the quantity $\delta > 0$ represents a constant that can be fixed arbitrarily small.

3.1. Simple version of the deterministic algorithm

We assume given integers $\pi_i \geqslant 2$ such that $q-1 = \rho\,\pi_1\cdots\pi_m$. We let $\chi = \pi_1\cdots\pi_m$, and $\sigma = \pi_1+\cdots+\pi_m$ as previously. The core algorithm of this section computes a sequence of Graeffe transforms, and then deduces inductively their roots starting from the last transform and finishing to the input polynomial.

Algorithm 13.

Input. $f \in \mathbb{F}_q[x]$ of degree d, monic, separable, which splits over $\mathbb{F}_q$, and such that $f(0) \neq 0$; a primitive root ξ of unity of order χ.

Output. $e_1,\ldots,e_s$ in $\{0,\ldots,\chi-1\}$, such that f divides $\prod_{i=1}^s (x^\rho - \xi^{e_i})$.

1. Compute the Graeffe transform h_0 of order ρ of f, and recursively compute h_i as the Graeffe transform of order π_i of h_{i-1}, for all $1 \leqslant i \leqslant m-1$.
2. Initialize E with the list $[0]$.
3. For i from m down to 1 do
   a. Replace E by the concatenation of $[(e+j\,\chi)/\pi_i \,|\, e \in E]$ for j from 0 to $\pi_i - 1$.

b. If E has cardinality more than d then remove the elements e from E such that $h_{i-1}(\xi^e) \neq 0$, using a multi-point evaluation to compute the $h_{i-1}(\xi^e)$.

4. Return E.

Proposition 14. *Algorithm 13 is correct. If $d^{1+\delta} \leqslant q-1$ then it costs $\sigma\, O(\mathsf{M}_q(d)\log d) + (d\log^2 q)^{1+\delta} + O(m\, d\, \mathsf{I}(\log\chi) + m\, \mathsf{m}_q\, d\log\chi) = \sigma\,\tilde{O}(d\log q) + (d\log^2 q)^{1+\delta} + \tilde{O}(d\log^3 q)$. Whenever a primitive root of unity of order ρ is given, and $\mu := \max(\rho, \pi_1, ..., \pi_m) = O(\log q)$, this bound further reduces to $\tilde{O}(d\log^3 q)$.*

Proof. We prove the correctness by descending induction on i from m down to 1. At the end of iteration i of the loop of step 3, we claim that f divides $\prod_{e\in E}(x^{\rho\pi_1\cdots\pi_{i-1}} - \xi^e)$, and that the integers e in E are in $\{0, ..., \chi-1\}$ and are divisible by $\rho\,\pi_1\cdots\pi_{i-1}$. These properties all hold for $i = m+1$ when entering step 3. By induction, we may assume that these properties hold with $i+1$ when entering the loop at level i. Let $e \in E$, and let α be a root of f such that $\alpha^{\rho\pi_1\cdots\pi_i} = \xi^e$. Since e is divisible by π_i, there exists $j \in \{0, ..., \pi_i - 1\}$ such that $\alpha^{\rho\pi_1\cdots\pi_{i-1}} = \xi^{(e+j\chi)/\pi_i}$. This proves the correctness.

From now on we assume that $d^{1+\delta} \leqslant q-1$. As to the complexity, we can compute the χ/π_i for $i \in \{1, ..., m\}$ in time $O(m\,\mathsf{I}(\log\chi))$ as follows: we first compute π_1, $\pi_1\,\pi_2$, $\pi_1\,\pi_2\,\pi_3$, ..., $\pi_1\,\pi_2\cdots\pi_{m-1}$, and π_m, $\pi_{m-1}\,\pi_m$, $\pi_{m-2}\,\pi_{m-1}\,\pi_m$, ..., $\pi_2\cdots\pi_{m-1}\,\pi_m$, and then deduce each χ/π_i by multiplying $\pi_1\cdots\pi_{i-1}$ and $\pi_{i+1}\cdots\pi_m$.

Step 1 requires time $(d\log^2 q)^{1+\delta}$ by Corollary 9. In step 3.a, we compute all the e/π_i in time $O(d\,\mathsf{I}(\log\chi))$, and then all the ξ^{e/π_i} and ξ^{χ/π_i} by means of $O(d\log\chi)$ operations in $\mathbb{F}_q$. Deducing all the $\xi^{(e+j\chi)/\pi_i}$ takes $O(\pi_i\, d)$ additional products in $\mathbb{F}_q$. The multi-point evaluations can be done in time $O(\pi_i\,\mathsf{M}_q(d)\log d)$.

Finally, when $\mu = O(\log q)$, the term $(d\log^2 q)^{1+\delta}$ can be replaced by $(\rho+\sigma)\,\tilde{O}(d\log q) = \tilde{O}(\sigma\, d\log q + d\log^2 q) = \tilde{O}(d\log^3 q)$ in view of Proposition 1. □

The cost in the latter proposition is asymptotically higher than the one of Mignotte–Schnorr's algorithm [20, Section 2], because of the term $(d\log^2 q)^{1+\delta}$. In the smooth case when $\mu = O(\log q)$, the asymptotic complexity bounds become similar.

The rest of this section is devoted to improve Algorithm 13. In order to diminish the dependence on $\log q$, we stop representing roots by their logarithms as in Mignotte–Schnorr's algorithm. Instead we shall compute discrete logarithms only when necessary. Furthermore, we will show how to decrease the dependence on the π_i to $\sqrt{\pi_i}$, by using the "baby-step giant-step" technique.

3.2. Reverse splitting of Graeffe transforms

Let g be a monic polynomial which splits over $\mathbb{F}_q$. Once the roots $a_1, ..., a_l$ of the Graeffe transform of order π of g have been recovered, we may decompose g into l factors $g_1, ..., g_l$ such that the π-th powers of the roots of g_i all coincide to a_i, for all $i \in \{1, ..., l\}$. In short, we have $\mathrm{Gr}_\pi(g_i)(x) \backsim (x-a_i)^{\deg g_i}$. This decomposition may be achieved efficiently *via* the following divide and conquer algorithm.

Algorithm 15.

Input. $g \in \mathbb{F}_q[x]$ of degree d, monic, separable, which splits over $\mathbb{F}_q$, and such that $g(0) \neq 0$; the pairwise distinct roots a_1, ..., a_l of the Graeffe transform of a given order $\pi \geqslant 2$ of g.

Output. The sequence of the monic $\gcd(x^\pi - a_i, g(x))$ for all $i \in \{1, ..., l\}$.

1. If $l = 1$ then return g.
2. Let $h := \lfloor l/2 \rfloor$.

3. Compute $e(x) := \prod_{i=1}^{h} (x - a_i)$.
4. Compute $g_1(x) := \gcd(e(x^\pi), g(x))$ and $g_2(x) := g(x) / g_1(x)$.
5. Call recursively the algorithm with input $g_1, a_1, ..., a_h$ and $g_2, a_{h+1}, ..., a_l$, and return the concatenation of the polynomial sequences returned so.

PROPOSITION 16. *Algorithm 15 is correct and costs* $\pi\, \tilde{O}(d \log q \log \pi)$ *or* $(d \log q)^{1+\delta} + \tilde{O}(d \log \pi \log q)$, *for any constant* $\delta > 0$.

Proof. If $l = 1$, then the π-th powers of the roots of g are all equal to a_1, whence the correctness. If $l > 1$, then the roots of g_1 are exactly those of g whose π-th powers are in the set $\{a_1, ..., a_h\}$. Consequently, the π-th powers of the roots of g_2 are all in the set $\{a_{h+1}, ..., a_l\}$. This completes the proof of the correctness by induction.

Step 3 can be performed in softly linear time by the subproduct tree technique (for the sake of efficiency the subproduct tree should be shared between all the recursive calls).

On the one hand, computing g_1 can be done naively in time $O(\mathsf{M}(\pi\, d) + \mathsf{M}(d) \log d) = \pi\ \tilde{O}(d\ \log\ q\ \log\ \pi)$. On the other hand, one may proceed as follows: compute $r(x) := x^\pi \operatorname{rem} g(x)$ in time $\tilde{O}(d \log \pi \log q)$, and then use [27, Corollary 7.2, p. 1789] to obtain $e(r(x)) \operatorname{rem} g(x)$ in time $(d \log q)^{1+\delta}$.

The sum of the degrees of the input polynomials g at the same depth of the recursive calls is at most d, and the maximal depth is $O(\log l) = O(\log d)$. This yields the complexity bounds of the algorithm. □

3.3. Finding roots of several polynomials in a given set

Let $g_1, ..., g_l$ be polynomials of respective degrees $d_1, ..., d_l$, and let $d := d_1 + \cdots + d_l$. For given subset $\mathcal{A}$ of $\mathbb{F}_q$ of cardinality $\leqslant d$, we are interested in finding the roots of g_i in $\mathcal{A}$ for each $i \in \{1, ..., l\}$ simultaneously. We do not make additional assumptions, and in particular the g_i are allowed to share common roots. A natural divide and conquer approach applies, as explained in the following algorithm.

Algorithm 17.

Input. $g_1, ..., g_l \in \mathbb{F}_q[x]$ of degrees $\geqslant 1$; a subset $\mathcal{A}$ of $\mathbb{F}_q$ of cardinality at most $d := \Sigma_{i=1}^{l} \deg g_i$.

Output. The sequence of the sets of the roots of $g_1, ..., g_l$ in $\mathcal{A}$.

1. If $l = 1$, then evaluate g_1 at all the points of $\mathcal{A}$, and return those which are roots.
2. Let $h := \lfloor l/2 \rfloor$.
3. Compute $e_1 := \prod_{i=1}^{h} g_i$ and $e_2 := \prod_{i=h+1}^{l} g_i$.
4. Compute the subset $\mathcal{A}_1$ of points of $\mathcal{A}$ which are roots of e_1.
5. Compute the subset $\mathcal{A}_2$ of points of $\mathcal{A}$ which are roots of e_2.
6. Call recursively the algorithm with input g_1, ..., g_h, $\mathcal{A}_1$ and g_{h+1}, ..., g_l, $\mathcal{A}_2$, and return the concatenation of the returned sequences of sets of roots.

PROPOSITION 18. *Algorithm 17 is correct and costs* $O(\mathsf{M}_q(d) \log d \log l) = \tilde{O}(d \log q)$.

Proof. For the correctness, it is clear that the roots of g_1, ..., g_h in $\mathcal{A}$ are to be found in $\mathcal{A}_1$ — idem for $\mathcal{A}_2$. The recursive calls are legitimate since the cardinality of $\mathcal{A}_1$ (resp. of $\mathcal{A}_2$) cannot exceed $\deg g_1$ (resp. $\deg g_2$).

Steps 1 to 5 take $O(\mathsf{M}_q(d) \log d)$. The sum of the degrees of the input polynomials at the same depth of the recursive calls is at most d, and the maximal depth is $O(\log l)$. This yields the complexity bound, thanks to the super-additivity of M_q. □

Notice that this lemma generalizes to arbitrary fields.

3.4. Simultaneous baby-step giant-step root finding

Instead of using multi-point evaluations in step 3.b of Algorithm 13, we may split the separable part of h_{i-1} thanks to Algorithm 15. We will detail later how this situation further reduces to computing all the roots of several polynomials of degree sum at most d, and having roots of order dividing π. To perform efficient root finding in this case, we adapt Pollard–Strassen's aforementioned strategy for integer factoring as follows.

Algorithm 19.

Input. $g_1, ..., g_l \in \mathbb{F}_q[x]$ of degrees $\geqslant 1$, monic, separable, which split over $\mathbb{F}_q$, and such that $g_j(0) \neq 0$ holds for all $j \in \{1, ..., l\}$; a multiple π of the orders of the roots of all the g_j; a primitive root ξ of unity of order π.

Output. The sets L_j of the ξ-logarithms of the roots of g_j for all $j \in \{1, ..., l\}$.

1. Let $d := \sum_{j=1}^{l} \deg g_j$, let $s := \lfloor \sqrt{\pi\, d} \rfloor$, and $t := \lceil \pi / s \rceil$.
2. Compute $h(x) := \prod_{j=0}^{s-1} (\xi^j x - 1)$.
3. For all $j \in \{1, ..., l\}$ and all $i \in \{0, ..., t-1\}$, compute $r_{j,i}(x) := h(x) \operatorname{rem} g_j(\xi^{si} x)$.
4. For all $j \in \{1, ..., l\}$, compute the set I_j of indices $i \in \{0, ..., t-1\}$ such that $r_{j,i}(x)$ has a proper gcd $e_{j,i}(x)$ with $g_j(\xi^{si} x)$.
5. For all $j \in \{1, ..., l\}$ and all $i \in I_j$, compute all the ξ-logarithms $E_{j,i}$ of the roots of $e_{j,i}$ in $\mathcal{A} = \{1, \xi, \xi^2, ..., \xi^{s-1}\}$ by calling Algorithm 17 $\lceil s/d \rceil$ times on subsets of $\mathcal{A}$ of cardinalities at most d.
6. Return the sets $L_j := \bigcup_{i \in I_j} (s\, i + E_{j,i})$, for all $j \in \{1, ..., l\}$.

Proposition 20. *Algorithm 19 is correct and costs* $\tilde{O}\big(\sqrt{\pi\, d} \log q\big) + \tilde{O}(d \log q)$.

Proof. Of course, the returned values in L_j are ξ-logarithms of roots of g_j. Conversely, let ξ^u be a root of g_j with $0 \leqslant u \leqslant \pi - 1$. There exists a unique i in $\{0, ..., t-1\}$ such that $i\, s \leqslant u < (i+1)\, s$, since $s\, t \geqslant \pi$. Then ξ^{u-is} is a common root of $g_j(\xi^{si} x)$ and $h(x)$. This proves the correctness.

By Lemma 2, step 2 executes in time $O(\mathsf{M}_q(s)) = \tilde{O}\big(\sqrt{\pi\, d} \log q\big)$. Since $d\, t = s + O(d)$, step 3 costs

$$O(\mathsf{M}_q(s) \log(s) + \mathsf{M}_q(d) \log d) = \tilde{O}\big(\sqrt{\pi\, d} \log q\big) + \tilde{O}(d \log q).$$

Using the super-additivity of M_q, step 4 needs $O(t\ \mathsf{M}_q(d)\ \log\ d) = \tilde{O}\big(\sqrt{\pi\, d}\ \log\ q\big) + \tilde{O}(d \log q)$. Since the sum of the degrees of all the polynomials $e_{j,i}$ is at most d, Proposition 18 implies a cost $\tilde{O}(s \log q + d \log q)$ for step 5. □

If $l = 1$, then Algorithm 19 slightly improves Shoup's variant of Pollard–Strassen's algorithm described in [43], which takes $s = \sqrt{\pi}$ independently of d; taking $s = \sqrt{\pi\, d}$ turns out to be a better trade-off between the baby steps and the giant steps. We also notice that this trade-off might be further improved by introducing logarithm factors in s.

3.5. Fast discrete logarithm in smooth finite fields

Let us recall that the term $\tilde{O}(d \log^3 q)$ in the complexity of Algoritm 13 is due to computing roots from their logarithms. In the next subsection we will proceed differently and will compute logarithms only when needed. This is why we now address the discrete logarithm problem. Let us recall that the seminal discrete logarithm method in the smooth case of $\mathbb{F}_p^*$ goes back to [36] and performs $\sqrt{S_1(p-1)}\, O(\log p)$ operations in $\mathbb{F}_p$, making use internally of Shanks' baby-step giant-step paradigm. We recall this algorithm for completeness, adapting it to our context of Turing machines.

Algorithm 21.

Input. A primitive root $\xi \in \mathbb{F}_q^*$ of order $\pi \geqslant 2$ of unity; a in the multiplicative subgroup generated by ξ.

Output. The ξ-logarithm of a in $\{0, ..., \pi - 1\}$.

1. Let $s := \lfloor \sqrt{\pi} \rfloor$ and $t := \lceil \pi / s \rceil$.
2. Compute $1, \xi, \xi^2, ..., \xi^{s-1}$.
3. Compute $a, a\,\xi^{-s}, a\,\xi^{-2s}, ..., a\,\xi^{-(t-1)s}$.
4. Sort the array made of the pairs $(i,\ \xi^i)$ for $i \in \{0,\ ...,\ s-1\}$ and $(j,\ a\ \xi^{-js})$ for $j \in \{0, ..., t-1\}$, accordingly to the lexicographic order of the second entries of the pairs seen as k-tuples in $\{0, ..., p-1\}^k$.
5. Find the first two consecutive pairs (e_1, ξ^{e_1}) and $(e_2, a\,\xi^{-e_2 s})$ whose second entries coincide.
6. Return $e_1 + s\,e_2 \operatorname{rem} \pi$.

PROPOSITION 22. *Algorithm 21 is correct and requires a running time* $\sqrt{\pi}\ \tilde{O}(\log\ q) + \sqrt{\pi} \log \pi\, O(\log q)$.

Proof. Since $s\ t \geqslant \pi$, the discrete logarithm $e = \log_\xi a$ can be uniquely written as $e = e_2\, s + e_1$ with $0 \leqslant e_1 < s$ and $0 \leqslant e_2 < t$. This proves the correctness.

Steps 2 and 3 perform $O(\sqrt{\pi})$ products in $\mathbb{F}_q$ and one inversion. Using the merge-sort algorithm, step 4 costs $O(\sqrt{\pi} \log q \log \pi)$. □

The following divide and conquer approach allows us to compute logarithms for composite orders $\chi = \pi_1 \cdots \pi_m$. Independently from the previous algorithm, it leads to a good complexity bound in terms of $\sigma = \pi_1 + \cdots + \pi_m$. This approach might already be known, but we could not find a reference to it in the literature.

Algorithm 23.

Input. $\xi \in \mathbb{F}_q^*$ of order $\chi = \pi_1 \cdots \pi_m$, with given $\pi_i \geqslant 2$; a in the multiplicative subgroup generated by ξ.

Output. The ξ-logarithm of a in $\{0, ..., \chi - 1\}$.

1. If $m = 1$, then find and return $e \in \{0, ..., \pi_1 - 1\}$ such that $a = \xi^e$.
2. Let $m_1 := \lfloor m/2 \rfloor$, $m_2 := m - m_1$, $\chi_1 := \pi_1 \cdots \pi_{m_1}$ and $\chi_2 := \chi / \chi_1$.
3. Recursively compute $e_1 := \log_{\xi^{\chi_1}} a^{\chi_1}$.
4. Recursively compute $e_2 := \log_{\xi^{\chi_2}} (a / \xi^{e_1})$.
5. Return $e_1 + \chi_2\, e_2$.

PROPOSITION 24. *Algorithm 23 is correct and executes in time* $O(\sigma\ \mathsf{m}_q + \mathsf{d}_q \log\ \chi + \mathsf{m}_q \log\chi \log\log\chi + \mathsf{I}(\log\chi) \log\log\chi) = \sigma\, \tilde{O}(\log q) + \tilde{O}(\log q \log \chi)$. *By using Algorithm 21 in step 1, the cost becomes* $\sqrt{\mu}\, \tilde{O}(\log^2 q)$, *where* $\mu := \max(\pi_1, ..., \pi_m)$.

Proof. We prove the correctness by induction. The case $m = 1$ is clear. Assume $m \geqslant 2$. In step 3, we have $e_1 \in \{0, ..., \chi_2 - 1\}$. By definition of e_1, we have $(a / \xi^{e_1})^{\chi_1} = a^{\chi_1} / \xi^{\chi_1 e_1} = 1$ in step 4, whence a / ξ^{e_1} indeed lies in the multiplicative subgroup of order χ_1 generated by ξ^{χ_2}. By definition of e_2, we have $a / \xi^{e_1} = (\xi^{\chi_2})^{e_2}$, whence $a = \xi^{e_1 + \chi_2 e_2}$, which completes the induction.

As to the complexity bound, let $T(\chi)$ denote the cost of the algorithm. We share the subproduct tree built from $\pi_1, ..., \pi_m$, in time $O(\mathsf{I}(\log \chi) \log m)$, between the recursive calls. Using binary powering in steps 3 and 4, we obtain the recursion relation

$$T(\chi)\ \leqslant\ O(\mathsf{m}_q \log \chi + \mathsf{d}_q) + T(\chi_1) + T(\chi_2).$$

which leads to the bound $T(\chi) \leqslant \Sigma_{i=1}^{m} T(\pi_i) + O(m\,\mathsf{d}_q + \mathsf{m}_q \log\chi \log m)$.

Using a naive exhaustive search in step 1 yields $T(\pi_i) = \pi_i\,\mathsf{m}_q$, whence $T(\chi) = O(\sigma\,\mathsf{m}_q + m\,\mathsf{d}_q + \mathsf{m}_q \log\chi \log m + \mathsf{I}(\log\chi)\log m)$. The first bound thus follows using $m = O(\log\chi)$. The second bound follows from Proposition 22 that gives $T(\pi_i) = \sqrt{\pi_i}\log\pi_i\,\tilde{O}(\log q)$. □

Remark 25. As a byproduct of the complexity analysis, we remark that, given a primitive root of unity ξ of order $\chi = \pi_1 \cdots \pi_m$, where the π_i are given integers $\geqslant 2$, one may compute primitive roots of unity ξ_i for all orders π_i in time $\tilde{O}(\log q \log\chi)$.

3.6. Squarefree part

Over any field $\mathbb{K}$ the separable factorization of a univariate polynomial g of degree d can computed by means of $O(\mathsf{M}(d)\log d)$ operations in $\mathbb{K}$ [29]. If $\mathbb{K}$ is the finite field $\mathbb{F}_q$ then it is possible to deduce the squarefree factorization of g with $O(d)$ additional extractions of p-th roots [29, Section 4]. This result is also left as an exercise in [17, Chapter 14, Exercise 14.30]. In the special case when $k = 1$ and $p > d$, no root extraction is necessary and the squarefree part of g is obtained as $g \,/\, \gcd(g, g')$. In the next subsection we will need the following bound, taking the number of simple and multiple roots into account.

Proposition 26. *Let $g \in \mathbb{F}_q[x]$ be of degree d. In an algebraic closure of $\mathbb{F}_q$, let δ_1 be the number of simple roots of g, let $\delta_{\geqslant 2}$ be the number of its multiple roots, and let $\Delta_{\geqslant 2} = d - \delta_1$ be the number of multiple roots counted with multiplicities. Then the squarefree factorization and the squarefree part of f may be computed in time $\tilde{O}(d\log q) + \Delta_{\geqslant 2}\,\tilde{O}(\log^2 q)$.*

Proof. Let $(g_1, q_1, \nu_1), \ldots, (g_l, q_l, \nu_l)$ in $(\mathbb{F}_q[x] \setminus \mathbb{F}_q) \times \{1, p, p^2, \ldots,\} \times \mathbb{N}$ represent the separable factorization of g [29, Introduction], so that we have $g(x) = g_1(x^{q_1})^{\nu_1} \cdots g_l(x^{q_l})^{\nu_l}$. Since $\mathbb{F}_q$ is perfect, the factors $g_i(x^{q_i})$ with $q_i = \nu_i = 1$ contain all the simple roots of g. Letting $h_1 = \prod_{q_i = \nu_i = 1} g_i$ and $h_{\geqslant 2} = g \,/\, h_1$, we have $\deg h_1 = \delta_1$ and $\deg h_{\geqslant 2} = \Delta_{\geqslant 2}$. Following [29, Section 4], the actual number of p-th root extractions is $O(\Delta_{\geqslant 2})$. Each such extraction amounts to $O(\log q)$ products in $\mathbb{F}_q$. In this way we obtain the squarefree factorization of g. □

3.7. Improved root finding algorithm

We are now ready to combine all the previous algorithms of this section in order to improve Algorithm 13. Recall that we are given integers $\pi_i \geqslant 2$ such that $q - 1 = \rho\,\pi_1 \cdots \pi_m$, and that we let $\chi = \pi_1 \cdots \pi_m$. The following algorithm is parametrized by subroutines used internally. We distinguish three cases of use, leading to complexity bounds in terms of σ, of μ, and in the smooth case of $S_1(q-1) = O(1)$.

Algorithm 27.

Input. $f \in \mathbb{F}_q[x]$, monic, separable, which splits over $\mathbb{F}_q$, of degree $d < q$, such that $f(0) \neq 0$; a primitive root ζ of $\mathbb{F}_q^*$.

Output. The roots of $\mathrm{Gr}_\rho(f)$.

1. Compute h_0 as the Graeffe transform of f of order ρ. Compute $\tilde{h}_0$ as the squarefree part of h_0. Recursively compute h_i as the Graeffe transform of order π_i of $\tilde{h}_{i-1}$, and $\tilde{h}_i$ as the squarefree part of h_i, for all i from 1 to $m-1$.
2. Initialize Z with $\{1\}$, and compute $\xi = \zeta^\rho$.
3. For i from m down to 1 do
   a. Use Algorithm 15 with input $\tilde{h}_{i-1}$ and $Z = \{a_1, \ldots, a_l\}$, and let $g_1, \ldots, g_l$ be the polynomials obtained in return.

b. Initialize W_0 with the roots of all the g_j of degree 1, for $j \in \{1, ..., l\}$.
c. For all $j \in \{1, ..., l\}$ such that $\deg g_j \geqslant 2$, compute the ξ-logarithm e_j of a_j with Algorithm 23 (resp. with Algorithms 23 and 21).
d. Use Algorithm 17 (resp. Algorithm 19) to compute the sets of roots W_j of $g_j\big(\xi^{e_j/\pi_i} x\big)$ for all $j \in \{1, ..., l\}$ such that $\deg g_j \geqslant 2$.
e. $Z := W_0 \cup \bigcup_j \big(\xi^{e_j/\pi_i} W_j\big)$.

4. Return Z.

PROPOSITION 28. *Algorithm 27 is correct and costs* $(\sigma + \rho)\, \tilde{O}(d \log q \log(\mu\, \rho)) + \tilde{O}(d \log^2 q)$. *In addition, for any fixed* $\delta > 0$, *the cost is also bounded by* $\sigma\, \tilde{O}(d \log q) + (d \log^2 q)^{1+\delta}$ *or* $\tilde{O}(\sqrt{\mu}\ d \log^2 q) + (d \log^2 q)^{1+\delta}$, *whenever* $d^{1+\delta} \leqslant q - 1$ *and where* $\mu = \max(\pi_1, ..., \pi_m)$.

Proof. Let introduce $h_m := \mathrm{Gr}_{\pi_m}\big(\tilde{h}_{m-1}\big)$, which does not need to be computed. When entering the loop in step 3, the set Z contains the single root of h_m. We prove by descending induction on i from m down to 1 that Z contains the roots of h_i when entering step i of the loop. Assuming this hypothesis holds for $i \leqslant m$, conditions of Algorithm 15 are met, and finding the roots of $\tilde{h}_{i-1}$ reduces to finding the roots of g_1, ..., g_l. The g_j of degree 1 contribute to roots of $\tilde{h}_{i-1}$ in a straightforward manner. For the other g_j, of degree at least 2, it turns out that the roots of $g_j\big(\xi^{e_j/\pi_i} x\big)$ have order dividing π_i, and thus Algorithms 17 or 19 produce the remaining roots of $\tilde{h}_{i-1}$. This shows the correctness.

First all, the primitive roots of unity of orders π_i and ρ needed during the execution of the algorithm can be obtained in time $\tilde{O}(\log^2 q)$ by Remark 25. The Graeffe transforms in step 1 may execute in time $(\sigma + \rho)\, \tilde{O}(d \log q \log(\mu\, \rho))$ by Proposition 1.

Let $d_i := \deg h_i$, let $\delta_{1,i}$ be the number of simple roots of h_i, let $\delta_{\geqslant 2,i}$ be the number of multiple roots of h_i, and let $\Delta_{\geqslant 2,i} = d_i - \delta_{1,i}$ be the number of multiple roots of h_i counted with multiplicities. By Proposition 26, computing the squarefree part $\tilde{h}_i$ of h_i takes time $\tilde{O}(d \log q) + \Delta_{\geqslant 2,i}\, \tilde{O}(\log^2 q)$. From $\deg \tilde{h}_i = \delta_{1,i} + \delta_{\geqslant 2,i}$ and $\deg \tilde{h}_{i-1} = d_i$, we obtain $\deg \tilde{h}_{i-1} - \deg \tilde{h}_i = d_i - \delta_{1,i} - \delta_{\geqslant 2,i} = \Delta_{\geqslant 2,i} - \delta_{\geqslant 2,i}$. Summing these equalities over i leads to $\deg \tilde{h}_0 - \deg \tilde{h}_m = \sum_{i=1}^m \Delta_{\geqslant 2,i} - \sum_{i=1}^m \delta_{\geqslant 2,i}$. Using $\Delta_{\geqslant 2,i} \geqslant 2\, \delta_{\geqslant 2,i}$, $\deg \tilde{h}_0 = d$, and $\deg \tilde{h}_m = 1$, we deduce that $\sum_{i=1}^m \delta_{\geqslant 2,i} \leqslant d - 1$, and then that $\sum_{i=1}^m \Delta_{\geqslant 2,i} \leqslant 2\,(d-1)$. Consequently, the total cost for computing all the squarefree parts drops to $\tilde{O}(d \log^2 q)$.

Step 3.a may take $\pi_i\, \tilde{O}(d \log q \log \pi_i)$ by Proposition 16, which yields a total cost $\sigma\, \tilde{O}(d \log q \log \mu)$. In step 3.c, Algorithm 23 is called $O(\delta_{\geqslant 2,i})$ times. Consequently, the total cost of this step is $\sigma\, d\, \tilde{O}(\log q) + d\, \tilde{O}(\log^2 q)$ by Proposition 24. If we use Algorithm 17 in step 3.d, then the cost is $\lceil \pi_i / d \rceil\ \tilde{O}(d \log q) = \pi_i\, \tilde{O}\big(\log^{O(1)} d \log q\big) + \tilde{O}(d \log q)$ by Proposition 18. The sum of these costs is bounded by $\sigma\, \tilde{O}(d \log q)$. So far, this establishes the first complexity bound.

From now on assume that $d^{1+\delta} \leqslant q - 1$. The cost for the Graeffe transforms in step 1 may drop into $(d \log^2 q)^{1+\delta}$ by Corollary 9. Step 3.a may run in time $(d \log q)^{1+\delta} + \tilde{O}(d \log \pi_i \log q)$ by Proposition 16. The total cost of this step thus amounts to $(d \log^2 q)^{1+\delta}$. Putting these bounds together yields the second complexity.

For the third complexity bound, we use Algorithm 23 combined with Algorithm 21 in step 3.c, so that Proposition 24 ensures a time $\sqrt{\mu}\ d\ \tilde{O}(\log^2 q)$. In addition we use Algorithm 19 in step 3.d, and Proposition 20 gives a total cost $\tilde{O}(\sqrt{\mu\, d} \log^2 q) + \tilde{O}(d \log^2 q)$. □

Recall that $S_1(q-1)$ represents the largest prime number in the irreducible factorization of $q-1$. If $S_1(q-1) = O(\log q)$ and if $\sigma + \rho = O(\log q)$, then the first complexity bound of Proposition 28 rewrites into $\tilde{O}(d \log^2 q)$, which is thus softly equivalent to the average cost of Cantor–Zassenhaus' algorithm. We are now able to state the main result of this section.

Theorem 29. *Given a constant $\delta > 0$, the irreducible factorization of $q-1$, a primitive element of $\mathbb{F}_q^*$, and $f \in \mathbb{F}_q[x]$ of degree d, monic, separable, which splits over $\mathbb{F}_q$, the roots of f can be computed in time $\tilde{O}\big(\sqrt{S_1(q-1)}\, d \log^2 q\big) + (d \log^2 q)^{1+\delta}$.*

Proof. Without loss of generality we can assume that $f(0) \neq 0$ and that $d \leqslant q-1$. If $d^{1+\delta} > q-1$, then it suffices to evaluate f at all the elements of $\mathbb{F}_q^*$ in time $\tilde{O}(d+q) = O(d^{1+\delta})$. Consequently, for when $d^{1+\delta} \leqslant q-1$, we appeal to the preceding proposition. □

Let us finally mention that, in general, the present method improves [16, Theorem 4.1] (see complexity details at the end of the proof). Nevertheless, if $k=1$, and if $S_1(p-1)$ is close to p, then Theorem 29 does not improve [42, Theorem 4.1(2)].

Bibliography

[1] M. Arora, G. Ivanyos, M. Karpinski, and N. Saxena. Deterministic polynomial factoring and association schemes. *LMS J. Comput. Math.*, 17(1):123–140, 2014.

[2] E. Bach. Comments on search procedures for primitive roots. *Math. Comp.*, 66:1719–1727, 1997.

[3] E. R. Berlekamp. Factoring polynomials over finite fields. *Bell System Tech. J.*, 46:1853–1859, 1967.

[4] E. R. Berlekamp. Factoring polynomials over large finite fields. *Math. Comp.*, 24:713–735, 1970.

[5] A. Bostan, Ph. Flajolet, B. Salvy, and É. Schost. Fast computation of special resultants. *J. Symbolic Comput.*, 41(1):1–29, 2006.

[6] A. Bostan, P. Gaudry, and É. Schost. Linear recurrences with polynomial coefficients and application to integer factorization and Cartier-Manin operator. *SIAM J. Comput.*, 36(6):1777–1806, 2007.

[7] A. Bostan, L. Gonzales-Vega, H. Perdry, and É. Schost. Complexity issues on Newton sums of polynomials. Distributed in the digital proceedings of MEGA'05, 2005.

[8] A. Bostan and É. Schost. Polynomial evaluation and interpolation on special sets of points. *J. Complexity*, 21(4):420–446, 2005.

[9] P. Camion. A deterministic algorithm for factorizing polynomials of $\boldsymbol{F}_q[X]$. In *Combinatorial mathematics (Marseille-Luminy, 1981)*, volume 75 of *North-Holland Math. Stud.*, pages 149–157. North-Holland, Amsterdam, 1983.

[10] D. G. Cantor and E. Kaltofen. On fast multiplication of polynomials over arbitrary algebras. *Acta Infor.*, 28(7):693–701, 1991.

[11] D. G. Cantor and H. Zassenhaus. A new algorithm for factoring polynomials over finite fields. *Math. Comp.*, 36(154):587–592, 1981.

[12] X. Caruso, D. Roe, and T. Vaccon. Tracking p-adic precision. *LMS J. Comput. Math.*, 17:274–294, 2014. Special Issue A, Algorithmic Number Theory Symposium XI.

[13] E. Costa and D. Harvey. Faster deterministic integer factorization. *Math. Comp.*, 83(285):339–345, 2014.

[14] S. Evdokimov. Factorization of polynomials over finite fields in subexponential time under GRH. In *Algorithmic number theory (Ithaca, NY, 1994)*, volume 877 of *Lecture Notes in Comput. Sci.*, pages 209–219. Springer, Berlin, 1994.

[15] Shuhong Gao. On the deterministic complexity of factoring polynomials. *J. Symbolic Comput.*, 31(1–2):19 – 36, 2001.

[16] J. von zur Gathen. Factoring polynomials and primitive elements for special primes. *Theoret. Comput. Sci.*, 52(1-2):77–89, 1987.

[17] J. von zur Gathen and J. Gerhard. *Modern computer algebra*. Cambridge University Press, 2nd edition, 2003.

[18] J. von zur Gathen and D. Panario. Factoring polynomials over finite fields: A survey. *J. Symbolic Comput.*, 31(1–2):3–17, 2001.

[19] M. Giusti, G. Lecerf, and B. Salvy. A Gröbner free alternative for polynomial system solving. *J. Complexity*, 17(1):154–211, 2001.

[20] B. Grenet, J. van der Hoeven, and G. Lecerf. Randomized root finding over finite fields using tangent Graeffe transforms. Technical report, École polytechnique, 2015.

[21] D. Harvey, J. van der Hoeven, and G. Lecerf. Even faster integer multiplication. http://arxiv.org/abs/1407.3360, 2014.

[22] D. Harvey, J. van der Hoeven, and G. Lecerf. Faster polynomial multiplication over finite fields. http://arxiv.org/abs/1407.3361, 2014.

[23] J. van der Hoeven and G. Lecerf. Sparse polynomial interpolation in practice. *ACM Commun. Comput. Algebra*, 48(4), 2014. In section "ISSAC 2014 Software Presentations".

[24] Ming-Deh A. Huang. Generalized Riemann hypothesis and factoring polynomials over finite fields. *J. Algorithms*, 12(3):464–481, 1991.

[25] E. Kaltofen. Polynomial factorization: a success story. In Hoon Hong, editor, *ISSAC '03: Proceedings of the 2003 International Symposium on Symbolic and Algebraic Computation*, pages 3–4. ACM Press, 2003.

[26] K. S. Kedlaya and C. Umans. Fast modular composition in any characteristic. In A. Z. Broder et al., editors, *49th Annual IEEE Symposium on Foundations of Computer Science 2008 (FOCS '08)*, pages 146–155. IEEE, 2008.

[27] K. S. Kedlaya and C. Umans. Fast polynomial factorization and modular composition. *SIAM J. Comput.*, 40(6):1767–1802, 2011.

[28] L. Kronecker. Grundzüge einer arithmetischen Theorie der algebraischen Grössen. *J. reine angew. Math.*, 92:1–122, 1882.

[29] G. Lecerf. Fast separable factorization and applications. *Appl. Alg. Eng. Comm. Comp.*, 19(2):135–160, 2008.

[30] G. Malajovich and J. P. Zubelli. Tangent Graeffe iteration. *Numer. Math.*, 89(4):749–782, 2001.

[31] M. Mignotte and C. Schnorr. Calcul déterministe des racines d'un polynôme dans un corps fini. *C. R. Acad. Sci. Paris Sér. I Math.*, 306(12):467–472, 1988.

[32] R. T. Moenck. On the efficiency of algorithms for polynomial factoring. *Math. Comp.*, 31:235–250, 1977.

[33] G. L. Mullen and D. Panario. *Handbook of Finite Fields*. Discrete Mathematics and Its Applications. Chapman and Hall/CRC, 2013.

[34] V. Pan. Solving a polynomial equation: Some history and recent progress. *SIAM Rev.*, 39(2):187–220, 1997.

[35] V. Pan. New techniques for the computation of linear recurrence coefficients. *Finite Fields Appl.*, 6:93–118, 2000.

[36] S. C. Pohlig and M. E. Hellman. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance (corresp.). *IEEE Trans. Inform. Theory*, 24(1):106–110, 1978.

[37] L. Rónyai. Factoring polynomials modulo special primes. *Combinatorica*, 9(2):199–206, 1989.

[38] L. Rónyai. Galois groups and factoring polynomials over finite fields. *SIAM J. Disc. Math.*, 5(3):345–365, 1992.

[39] Chandan Saha. Factoring polynomials over finite fields using balance test. In S. Albers and P. Weil, editors, *25th International Symposium on Theoretical Aspects of Computer Science*, volume 1 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 609–620, Dagstuhl, Germany, 2008. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. http://drops.dagstuhl.de/opus/volltexte/2008/1323.

[40] R. Schoof. Elliptic curves over finite fields and the computation of square roots mod p. *Math. Comp.*, 44(170):483–494, 1985.

[41] V. Shoup. On the deterministic complexity of factoring polynomials over finite fields. *Inform. Process. Lett.*, 33(5):261–267, 1990.

[42] V. Shoup. A fast deterministic algorithm for factoring polynomials over finite fields of small characteristic. In S. M. Watt, editor, *ISSAC '91: Proceedings of the 1991 International Symposium on Symbolic and Algebraic Computation*, pages 14–21. ACM Press, 1991.

[43] V. Shoup. Smoothness and factoring polynomials over finite fields. *Inform. Process. Lett.*, 38(1):39–42, 1991.

[44] V. Shoup. Searching for primitive roots in finite fields. *Math. Comp.*, 58:369–380, 1992.

[45] V. Shoup. *NTL: A Library for doing Number Theory*, 2014. Software, version 8.0.0. http://www.shoup.net/ntl.

[46] B. Źrałek. Using partial smoothness of $p-1$ for factoring polynomials modulo p. *Math. Comp.*, 79:2353–2359, 2010.