Deterministic equation solving over finite fields

Proefschrift

ter verkrijging van
de graad van Doctor aan de Universiteit Leiden,
op gezag van de Rector Magnificus Dr. D. D. Breimer,
hoogleraar in de faculteit der Wiskunde en
Natuurwetenschappen en die der Geneeskunde,
volgens besluit van het College voor Promoties
te verdedigen op dinsdag 16 mei 2006
klokke 14.15 uur

door

Christiaan Evert van de Woestijne

geboren te Rotterdam
in 1975

Samenstelling van de promotiecommissie:

promotor:	prof. dr. H. W. Lenstra, Jr.
referent:	prof. dr. E. Bayer Fluckiger (École Polytechnique Fédérale de Lausanne)
overige leden:	dr. W. Bosma (Radboud Universiteit Nijmegen)
	prof. dr. R. J. F. Cramer (Universiteit Leiden/CWI, Amsterdam)
	prof. dr. P. Stevenhagen
	prof. dr. R. Tijdeman
	prof. dr. S. M. Verduyn Lunel

Thomas Stieltjes Institute
for Mathematics

Deterministic equation solving over finite fields

Mathematisch Instituut, Universiteit Leiden, The Netherlands
www.math.LeidenUniv.nl

ISBN-10: 90-9020668-X
ISBN-13: 978-90-9020668-4

Abstract

The present thesis is devoted to the solution of two closely related algorithmic representation problems. They read as follows.

Problem A1 Given a finite field $\mathbb{F}$, given a positive integer n and given nonzero elements $a_0, a_1, \ldots, a_n$ of $\mathbb{F}$, compute $x_0, x_1, \ldots, x_n \in \mathbb{F}$, not all zero, such that

$$\sum_{i=0}^{n} a_i x_i^n = 0.$$

Problem A2 Given a finite field $\mathbb{F}$, given a positive integer n, and given nonzero elements $b, a_1, \ldots, a_n$ of $\mathbb{F}$, compute $x_1, \ldots, x_n \in \mathbb{F}$ such that

$$\sum_{i=1}^{n} a_i x_i^n = b,$$

or show that no such exist.

These, respectively, are the problems of representing zero or nonzero elements by diagonal forms in many variables over finite fields. By the classical Chevalley-Warning theorem we know that the equation in Problem A1 is always solvable — see Chapter 2 for more details. In Problem A2, the restriction that *all elements of* $\mathbb{F}$ *be sums of* n*th powers of elements of* $\mathbb{F}$ ensures solvability of the equation in all cases. Chapter 2 presents the first proof, to my knowledge, of the fact that this condition is sufficient.

Now solvability results like the ones just mentioned are usually proved in ways that do not lead to efficient methods for actually finding solutions. The main result obtained here is the construction of two efficient algorithms, one for each problem, that really compute such solutions. The proof of this result will take up the entire thesis and will provide a detailed description and analysis of the two closely related algorithms.

Theorem A3 *There are deterministic algorithms for solving Problems A1 and A2 that take polynomial time in terms of* n *and* $\log q$*, where* q *is the cardinality of* $\mathbb{F}$*.*

Up to now, many efficient algorithms for solving nonlinear equations over finite fields have made essential use of probabilistic components. These randomised steps mostly serve to find a field element that lies outside a certain multiplicative subgroup of the field; for example, a non-square element. The novelty of the result presented here is to dispense with these randomised techniques: the methods of this thesis are purely deterministic.

The formulation of my algorithms is completely elementary, but involves several levels of subroutines. As auxiliary results of independent interest, it is shown, for a given n, how to generate a finite field over its prime field by an nth power, and how to write any given finite field element as a sum of nth powers, using at most n terms — or, in both cases, correctly to decide that these tasks are impossible (Chapters 4 and 5, respectively). Another cornerstone is a deterministic adaptation of the Tonelli-Shanks root taking algorithm, which is presented in detail in Chapter 3.

The description of my algorithms will show that they are practical in the stated version. The analysis of their running time, which is straightforward, shows that they perform only slightly worse than probabilistic methods for solving Problems A1 and A2.

Applications are given to several areas of computation over finite fields, which include computing a rational point on a quadric hypersurface, computing isomorphisms of quadratic spaces, computing elements or field generators of prescribed norm, trivialising central simple algebras of degree 2, and, surprisingly, computing rational points on elliptic curves.

An implementation in the computer algebra language Magma of all algorithms developed in this thesis is available from the author.

Contents

Chapter 1

Introduction

The theory of finite fields has played an important role in 20th century mathematics, in areas such as number theory, algebraic geometry, and combinatorics. The present thesis is a contribution to this theory that focuses on polynomial equations in many variables over finite fields, and how to achieve both efficiency and determinism in algorithms that compute solutions to such equations.

In this chapter, after a brief introduction to finite fields and their applications, we discuss three important guiding problems in the theory of equations over finite fields, which give a context to our results. We then give an overview of the thesis, and close with a paragraph of conventions and definitions.

1.1 Finite fields

The basic properties of finite fields may be found in textbooks on abstract algebra, such as [33, Section V.5]. An encyclopedic volume devoted entirely to finite fields is [36].

A finite field is simply a finite set $\mathbb{F}$, equipped with addition, subtraction, multiplication and division operations that satisfy the usual axioms of associativity and distributivity for such operations, such that addition is commutative, and such that every element except 0 has a multiplicative inverse. It can be proved that the multiplication on such a set must be commutative. The cardinality of $\mathbb{F}$ is always a power of some prime number, and up to isomorphism, there exists only one finite field with a given prime power cardinality.

For every prime number p, the set of residue classes of the integers modulo p is a field of p elements, which may be represented as $\{0, 1, \ldots, p-1\}$, or indeed as any set of p integers that are pairwise distinct modulo p. The fact that every nonzero residue class modulo a prime has a multiplicative inverse follows from the extended version of Euclid's algorithm for computing greatest common divisors, and is treated, for example, by Gauss in his *Disquisitiones arithmeticae* [24]. A field of prime cardinality p is called a *prime field* and is denoted by $\mathbb{F}_p$.

Galois fields. The prime fields are the easiest examples of finite fields, but they are not the only ones. For every power $q = p^e$ of p, there exists a finite field $\mathbb{F}_q$ having q elements, the structure of which is unique up to isomorphism, and that arises by adjoining a *formal zero* of an irreducible polynomial f to the prime field: if f is a polynomial with coefficients in $\mathbb{F}_p$ and is irreducible of degree e, then we have

$$\mathbb{F}_q \cong \mathbb{F}_p[X]/(f).$$

The notation GF(q) for a finite field of q elements (where GF stands for Galois Field) is used to honour the work of Galois, who introduced finite fields of prime power cardinality in an 1830 memoir [21]. His purpose here was quite different from ours: he found that several finite groups are conveniently described using both affine transformations on a finite field, considered as a vector space over its prime field, and automorphisms of such fields.

Another way to obtain non-prime fields is to reduce rings of algebraic integers modulo a *prime ideal*; this point of view was advanced by Dedekind in the 1870s. For example, if ζ denotes a primitive 5th root of unity, then

$$\mathbb{Z}[\zeta]/(3) \cong \mathbb{F}_{81}.$$

The ring $\mathbb{Z}[\zeta]$ can be described as the set of all elements of the form $a + b\zeta + c\zeta^2 + d\zeta^3$, for integers a, b, c, d; taking these modulo 3, we find that the elements of $\mathbb{F}_{81}$ have the same form, but with a, b, c, d running over the integers modulo 3. From the fact that the quotient ring is a field, we deduce that the ideal (3) is maximal in $\mathbb{Z}[\zeta]$, hence prime, and in particular, the element 3 is prime.

In this thesis, most of the time we will not have to distinguish between the cases where the base field is a prime field and where it is not. The exception is Chapter 5, but also the main results of that Chapter are formulated for arbitrary finite fields.

Applications of finite fields. As already indicated, finite fields are used in the classification of finite groups (see [26] for a recent treatment), and the development of algebraic number theory (see, for example, [16] or [18]).

Finite fields are of obvious significance when studying Diophantine equations, i.e., polynomial equations over the integers: we can often decide solvability of such equations by reducing the coefficients modulo some prime number p, and finding the possible residues of the solutions modulo p. For example, a prime number p cannot be represented by the *binary quadratic form* $X^2 + nY^2$, with n a positive integer, if the equation

$$x^2 + ny^2 \equiv 0 \pmod{p}$$

over the finite field $\mathbb{F}_p$ has only the trivial solution $(0, 0)$. The converse question, whether the existence of a solution over $\mathbb{F}_p$ is sufficient, is much more subtle; it was one of the main problems treated by Gauss in [24]. An introductory treatment is given in [19].

Taking a more geometric perspective, we consider the algebraic variety specified by some system of polynomial equations, and study the properties of the variety obtained by the reduction of the equations modulo a prime. The wish to extend the usual concepts of algebraic geometry to work also over fields of non-zero characteristic, such as finite fields, has led to important developments in this area. See also Sections 1.2.2 and 2.5.

Finite fields are finite sets with a special structure, and as such they are of much use in combinatorics. For example, their addition and multiplication tables are Latin squares — every row and every column contains every element exactly once. This has been used in the design of experiments, for example in biology, where such structures are desirable. Finite fields are also important in the classification of other combinatorial objects, such as Hadamard matrices, difference sets, and finite affine and projective geometries. The combinatorial applications are given briefly in [36] and in detail in [10].

Finally, practically important applications of finite fields have arisen in the field of communications technology: the problem of reliable transfer of data over a noisy channel was addressed by the theory of error-correcting codes, while the development of public key cryptography satisfies the need for secret communication using public information channels. For this, we refer to [31, 37].

1.2 Central questions

There are three central questions in the theory of equations over finite fields that are relevant for this thesis.

1. Decide if a given equation or system of equations over a given finite field is solvable, i.e., whether it has any solutions.

2. Compute the number of solutions to the equation, or provide upper and lower bounds for this number.

3. Actually compute one, several, or all solutions.

For a survey of what has been done on these central questions, and of the many different types of equations that have been considered, we refer the reader to Chapter 6 in [36], especially the end notes, and to Joly's survey article [30]. In the remainder of this section, we illustrate the first two basic questions by showing their relation to our own results. The third question takes up a section on its own, as this is where our main contribution lies.

1.2.1 Deciding solvability

By the example of binary quadratic forms given above, we already illustrated the *first question*, which deals with solvability of equations. Following the founding contributions of Fermat, Lagrange, and Gauss, many kinds of “higher congruences” were attacked in a more or less systematic way, for example, equations of the form

$$ax^k + by^k + cz^k = 0,$$

which arose in connection with Fermat's last theorem [36, p. 326]. In the 20th century however, quite general results were obtained regarding the solvability of certain polynomial equations over finite fields.

For example, we have the theorem of Chevalley and Warning of 1935 [15, 55], already mentioned above, from which the solvability of Problem A1 can be derived.

Theorem 1.1 *Let $\mathbb{F}$ be a finite field, and let f be a polynomial of degree n in v variables over $\mathbb{F}$. Let N be the number of solutions in $\mathbb{F}^v$ of the equation $f = 0$. If $v > n$, then N is divisible by the characteristic of $\mathbb{F}$.*

Corollary 1.2 *If, under the same assumptions as in the Theorem, we assume in addition that f is homogeneous, then the equation $f = 0$ has at least one nontrivial solution over $\mathbb{F}$.*

Here a *nontrivial solution* is a solution where not all variables are zero. The Corollary follows because we have $p > 1$ and because homogeneous polynomials always have the trivial zero $(0, 0, \ldots, 0)$. It is noteworthy that the solvability criterion given in this theorem depends only on the degree and the number of variables, and not on the coefficients. It also illustrates the general notion that equations become "easier" to solve if the number of variables increases.

This thesis is concerned with nontrivial zeros of *diagonal forms* over finite fields, as well as with the nonzero values represented by such forms (cf. the Abstract). Here a *form* of degree n means a homogeneous polynomial of total degree n (as in "quadratic form"); hence a diagonal form of degree n can be written as

$$\sum_{i=1}^{v} a_i X_i^n$$

for some nonnegative integer v, where we take the coefficients a_i to be nonzero.

It is seen that by the Corollary, a diagonal form f over a finite field $\mathbb{F}$ has a nontrivial zero whenever $v > n$; therefore our Problem A1 is always solvable, as it concerns a form of degree n in $n + 1$ variables. A proof of Corollary 1.2 for the case of diagonal forms is given in Section 2.4. In Chapter 6, we will transform this proof into an efficient *algorithm* for finding a solution to the equation $\sum_{i=0}^{n} a_i x_i^n = 0$ over a given finite field. Several other proofs of Corollary 1.2 exist (see for example [25, Theorem 2.3], [30, Chapitre 3], [33, Exer. IV.7]), but these do not lead to efficient methods for actually finding solutions to the equation. On the other hand, the proof we give works only for diagonal forms.

Problem A2 is a little more difficult to solve. Here we want a given form of degree n in n variables to represent a given nonzero element b, where both the coefficients and the element b are in a given finite field $\mathbb{F}$. By the Corollary, there certainly exist $x_1, \ldots, x_n, y \in \mathbb{F}$ such that

$$a_1 x_1^n + \ldots + a_n x_n^n + (-b) y^n = 0;$$

but the problem is that we cannot guarantee that $y \neq 0$ for at least one solution $(x_1, \ldots, x_n, y)$, and therefore this is not enough to solve Problem A2. An example where such an equation cannot be solved at all is

$$x^3 + y^3 + z^3 = \alpha$$

over the field $\mathbb{F}_4$ of 4 elements, where $\alpha \in \mathbb{F}_4$ is a generator for $\mathbb{F}_4$ over its subfield $\mathbb{F}_2$. Because the third power of every element in $\mathbb{F}_4$ is contained in $\mathbb{F}_2$, whereas α is not, the equation has no solutions.

In Theorem 2.3(ii) below we prove a sufficient condition for solvability: if every element of $\mathbb{F}$ can be written as a *sum of nth powers* of elements of $\mathbb{F}$, then the equation in Problem A2 is always solvable for this particular n and $\mathbb{F}$. This result is new; up to now, the weakest sufficient condition known, given by Schwarz in 1950 [44], was the requirement that $\gcd(n, q-1)$ be less than the characteristic p of $\mathbb{F}$. Moreover, in contrast to Schwarz's nonconstructive proof, the proof given below already has some structural similarities with our eventual algorithm 6.7 solving the diagonal equation, even if it does not quite lead to an algorithm itself.

We note that, by the efforts of many authors, the condition $n > v$ on the number of variables v (respectively $n \geq v$ for the inhomogeneous problem) has been considerably relaxed, in the case of diagonal forms. It is now known, for example, that every element of a finite field $\mathbb{F}$ of q elements can be written as a sum of nth powers with no more than $\lceil 32 \log n \rceil + 1$ terms, provided we have $q > n^2$ (see [59]). The proofs giving rise to these results, however, are generally nonconstructive.

1.2.2 Number of solutions

The *second central question* deals with the problem of determining the *number of solutions* of a given equation over a finite field. Specifically applied to equations defining algebraic curves, this problem has recently gained great importance in cryptography ("elliptic curve cryptography", see [31]) and the theory of error-correcting codes (the so-called Goppa codes, see [37]).

For example, whether a given elliptic curve is safe to use for encryption is determined for a large part by the number of points on the curve with coordinates in the (finite) base field. In 1985, Schoof published the first polynomial-time algorithm to determine this number (see [42]). As a by-product, he gave the first deterministic algorithm that can take at least some square roots in finite fields in polynomial time; we will come back to this in the next section. Made practical by additions of Elkies and Atkin and several others, the point-counting algorithm is now widely used. In the meantime, the same problem is being attacked at the moment for ever more complicated (systems of) equations, among others in the number theory group in Leiden (Edixhoven, Bröker, and previously Carls). Already for this subproblem many techniques of algebraic geometry are brought to action.

We note that the results of this thesis can be applied, not to point counting, but to the *construction* of points on elliptic curves over finite fields (see Section 7.6).

A number of very general results, known as the *Weil conjectures*, deal with the growth of the number of solutions of an equation as we extend the equation's base field (cf. the notes to this subject in [36] and the final chapters of [30]). As the name says, these were surmised by Weil, in [56], and proved after the Second World War by Weil, Dwork, and Deligne. The techniques needed for this proof gave rise to a spectacular development in the field of algebraic geometry, in the course of which the foundations of this field were newly laid; for this, mainly ideas of Serre and Grothendieck were important.

One part of the Weil conjectures, already proved in [56], is the following theorem, which gives upper and lower bounds for the number of solutions of diagonal forms over finite fields.

Theorem 1.3 *[56, p. 502] Let $\mathbb{F}$ be a finite field of cardinality q, and let n and v be positive integers. Let f be a diagonal form of degree n in v variables over $\mathbb{F}$. Then the number N of nontrivial zeros of f satisfies*

$$\left|N - q^{v-1}\right| \;\leq\; (n-1)^{v-1} q^{v/2}; \tag{1.4}$$

furthermore, for any nonzero $b \in \mathbb{F}$, the number N_b of solutions to the equation $f(x_1, \ldots, x_v) = b$ satisfies

$$\left|N_b - q^{v-1}\right| \leq (n-1)^v q^{(v-1)/2}. \tag{1.5}$$

We use this result to analyse the complexity of a simple *probabilistic* algorithm for solving Problems A1 and A2; see Section 2.5.

1.3 Algorithms for finding solutions

The third basic question given above asks for the *actual computation* of solutions to a system of equations over a finite field. This question will take most of our attention in this thesis.

1.3.1 Efficiency

The mere existence of a solution algorithm is obvious, as we can simply enumerate all possible values of the variables in the finite base field and see if they satisfy the equation.

However, except for very small cases, this *brute force* method takes too much time. Namely, for a field of q elements and an equation in v variables we must examine at worst q^v different possible zeros. As we can represent elements of such a field using about $O(\log q)$ bits, this means that the number of points to try grows exponentially in the number of bits needed to specify the equation (provided the equation does not have too many nonzero coefficients).

An algorithm is generally considered *efficient* if the number of operations it performs is bounded by a *polynomial in the bit length* of the input to the algorithm, and *inefficient* if the number of operations grows faster than any polynomial function of this bit length. Efficient algorithms in this definition are also commonly known as "polynomial-time" algorithms.

We therefore sharpen the basic question and ask for *efficient* algorithms that solve equations over finite fields. Specifically, we require the number of operations to grow at most polynomially with $\log q$ and also at most polynomially with the degree and the number of terms of the equation or system of equations.

As may be imagined, the activity of research into this topic grew enormously with the advent of fast computers in the 1960s. But also long before this time, authors like Legendre, Galois, and Gauss exhibited a vivid interest in methods that could reduce the time spent on tedious and error-prone pencil-and-paper computations. For example, these classical authors already knew the main ingredients of the Cantor-Zassenhaus algorithm for polynomial factorisation over finite fields: distinct degree factorisation, and equal degree factorisation by dividing up the roots into squares and nonsquares, translating the variable by a random element if necessary. See [22], Notes to Chapter 14, for several interesting references and quotations.

1.3.2 Currently known methods

Besides the obvious brute force method, we currently know the following methods for computing solutions to nonlinear equations over finite fields.

1. The algorithm of Tonelli-Shanks for taking square and higher roots.

2. Methods for univariate polynomial factorisation.

3. Schoof's algorithm for computing $\sqrt{a} \pmod{p}$ for a fixed $a \in \mathbb{Z}$ and varying primes p.

4. Probabilistic methods for equations in many variables based on the preceding univariate methods.

To this we might add elimination methods, such as Gaussian elimination, the use of resultants, and Gröbner bases, by which the solution of a system of equations is reduced to the solution of a single equation. All these methods, except Schoof's method, are standard techniques and are discussed in textbooks like [7], [16], or [22].

It is a remarkable fact that, for many problems in the theory of equations over finite fields, the only efficient algorithms known are *probabilistic.* All methods given above, again excepting Schoof's, are of *Monte Carlo* type: they always give correct answers, but are only *expected* to finish in polynomial time. (This is opposed to *Las Vegas* type algorithms, like Rabin's compositeness test for integers, which finish in polynomial time but where the results can be right or wrong subject to a certain probability distribution.)

Root taking. Root taking in finite fields is not always difficult; for example, if the field $\mathbb{F}$ has q elements and n is coprime to $q-1$, we compute an nth root of a for $a \in \mathbb{F}^*$ as a^v, where $v \in \mathbb{Z}$ is such that $vn \equiv 1 \pmod{q-1}$. The algebraic reason is that raising to the power n is an automorphism of the multiplicative group $\mathbb{F}^*$. The interesting case is therefore where n divides $q-1$.

The method of Tonelli-Shanks was found by Tonelli in 1891 [52]. It is group-theoretic in nature, and has been well known since Shanks's rediscovery of it (see [46], where it is only used to compute square roots). To compute the nth root of an element a in the finite field $\mathbb{F}$, it solves a discrete logarithm problem in the ℓ-Sylow subgroups of the multiplicative group $\mathbb{F}^*$, for all primes ℓ dividing n. For this purpose it uses generators of these subgroups; in other words, for each ℓ it needs an element of $\mathbb{F}^*$ that is not an ℓth power in $\mathbb{F}$. This makes the algorithm probabilistic, because the only efficient method known to find such elements is by guessing. (However, the Generalised Riemann Hypothesis implies that such elements can be found in polynomial time by just trying $1, 2, 3, \ldots$, if $\mathbb{F}$ is a prime field.)

We will use a deterministic adaptation of the Tonelli-Shanks method in our own algorithm; see Chapter 3.

The deterministic square root algorithm of Schoof is a remarkable application of his algorithm for computing the number of rational points on elliptic curves over finite fields, which was mentioned earlier. It is efficient (in the sense defined above) *only* if it is used to take a root of a *fixed* element $a \in \mathbb{Z}$, viewed as an element of various finite prime fields in which it is a square; the running time is polynomial in the size of these prime fields, but exponential in $\log |a|$. The algorithm could be used, for example, to compute $\sqrt{-1}$ efficiently in all prime fields where it exists (viz., in $\mathbb{F}_p$ where $p \equiv 1 \pmod 4$). See [42] for more details.

Multivariate methods. The relevance of root taking methods is for solving simple univariate and bivariate equations, such as

$$ax^n + b = 0 \quad \text{or} \quad ax^n + by^n = 0,$$

where it is enough to take an nth root of $-b/a$. Furthermore, we can solve diagonal equations like

$$a_1 x_1^n + \ldots + a_v x_v^n = 0$$

probabilistically by guessing values for x_2 up to x_v, and taking an nth root of

$$-(a_2 x_2^n + \ldots + a_v x_v^n)/a_1,$$

if possible. This is an example of the methods mentioned in the last item of the list given above; we analyse this method in more detail in Section 2.5, because if we assume $v = n + 1$, it is a probabilistic efficient algorithm to solve our Problem A1.

Polynomial factorisation. Many algorithms have been proposed for the factorisation of univariate polynomials over finite fields, beginning with the Berlekamp and the Cantor-Zassenhaus algorithms in the late 1960s. In 2000, it became at last feasible to factorise polynomials of degree over a million over the field of 2 elements [11]; such extreme applications require the use of highly nontrivial algorithms, the development of which still continues. In [22, Chapter 14], the reader will find a survey of current techniques.

As an illustration of the application of finite fields to Diophantine equations, current algorithms for polynomial factorisation over the integers (like the algorithm of Lenstra-Lenstra-Lovász [34], better known as LLL or L^3, and Van Hoeij's algorithm [29]) all depend on polynomial factorisation over finite fields.

Although in practice the factorisation problem for polynomials over finite fields can be considered as solved, several theoretical problems remain. The most important one is the question whether an efficient *deterministic* factorisation method exists. Not even the assumption of the Generalised Riemann Hypothesis is enough to guarantee this (cf. [22, p. 411]).

To end this section, we note that another well known square root algorithm, the Cipolla-Lehmer algorithm, is in fact a special case of the Cantor-Zassenhaus factorisation algorithm (see [4] for more details).

1.4 Overview of the thesis

After describing the field of study to which our work belongs, we now give an overview of the following chapters of this thesis.

In Chapter 2, we give the necessary background from finite field theory. Furthermore, we prove the solvability of Problems A1 and A2 by a constructive method due to Dem′yanov and Kneser, and show how these Problems can be solved by *probabilistic* algorithms. The relative performance of the probabilistic and deterministic algorithms given in this thesis is discussed in Section 7.2.

Chapters 3 to 6 describe the building blocks of the *deterministic* algorithms providing the proof of Theorem A3. It may be useful to remark that all existence claims about algoritms in this thesis, and Theorem A3 in particular, are proved *constructively*: explicit algorithms are presented that show the truth of these claims.

Chapter 3 describes the Tonelli-Shanks algorithm for root extraction in finite fields, and gives a deterministic variant of this algorithm that we use extensively. Its main result is the following.

Theorem 1.6 *There exists a deterministic algorithm that, given a finite field $\mathbb{F}$ with q elements, a positive integer n, and $n+1$ nonzero elements $a_0, \dots, a_n$ of $\mathbb{F}$, determines integers i and j and an element $\beta \in \mathbb{F}$ such that $0 \leq i < j \leq n$ and*

$$a_i/a_j = \beta^n,$$

in time polynomial in n and $\log q$.

The Theorem says actually that, given $n+1$ elements in $\mathbb{F}$, we can compute an nth root of the quotient of two among them; however, without close analysis of the elements a_i one cannot predict which two. Because of this selection feature, we call the method Selective Root Extraction.

Chapter 4 describes how to find an nth power generator for a given extension of finite fields. It also proves the following auxiliary result, which may be considered as a multiplicative version of the primitive element theorem for separable field extensions.

Theorem 1.7 *There exists a deterministic algorithm that, given finite fields $\mathbb{E}$ and $\mathbb{F}$, with $\mathbb{E} \subseteq \mathbb{F}$, and nonzero elements $\alpha_1, \ldots, \alpha_t$ of $\mathbb{F}$, computes integers $x_1, \ldots, x_t$ such that*

$$\prod_{i=1}^{t} \alpha_i^{x_i} \text{ generates the field } \mathbb{E}(\alpha_1, \ldots, \alpha_t) \text{ over } \mathbb{E},$$

and uses time polynomial in t and $\log |\mathbb{F}|$.

The main result of Chapter 4 is as follows.

Theorem 1.8 *There exists a deterministic algorithm which, given a finite field K of q elements and a positive integer n dividing $q-1$, computes $\beta \in K$ such that β^n generates K over its prime field, or decides that no such β exists, in time polynomial in n and $\log q$.*

In Chapter 5 we complete the proof of Theorem A3 for the case that all a_i equal 1 — in other words, we show how to write finite field elements as sums of nth powers for given n, where the sums have at most n terms. More specifically, we prove Theorem 1.9, showing that in this case the running time of our algorithms is almost equal to that of probabilistic methods (cf. Section 2.6).

Theorem 1.9 *There exists a deterministic algorithm that, given a finite field $\mathbb{F}$ with q elements and a positive integer n, finds $x_0, \ldots, x_n \in \mathbb{F}$, with $x_0 \neq 0$, such that $x_0^n + \ldots + x_n^n = 0$ in $\mathbb{F}$, and takes time $\tilde{O}(n^2 (\log q)^2)$ to finish.*

(See below for a definition of the $\tilde{O}$-notation.) The proof makes use of the algorithms of Chapters 3 and 4; notably, let $\mathbb{F}$ be a finite field of characteristic p and extension degree e over $\mathbb{F}_p$. For a nonzero element $a \in \mathbb{F}$, and a positive integer n, we first compute β such that $\mathbb{F} = \mathbb{F}_p(\beta^n)$; it follows that

$$a = \sum_{i=0}^{e-1} a_i (\beta^{i-1})^n,$$

where the a_i, which are in the prime field $\mathbb{F}_p$, are considered as integers between 0 and $p-1$. This gives a as a sum of nth powers with at most $(p-1) \cdot e$ terms; the second stage of the algorithm reduces this to n via a logarithmic decay of the number of terms.

The case of arbitrary coefficients a_i is treated in Chapter 6, in which the proof of Theorem A3 is completed. The resulting algorithm uses the algorithms of Chapters 3, 4, and 5 as subroutines, and has similarities to the constructive method of Dem′yanov and Kneser referred to above. Because it maintains a system of equations whose shape is trapezoidal, we called it the *trapezium method.* It comes in two "flavours", homogeneous and inhomogeneous, corresponding to Problems A1 and A2, respectively.

Finally, Chapter 7 gives several applications and generalisations of the results in this thesis, as well as a performance comparison between our deterministic methods and the probabilistic method discussed in Section 2.6.

The most important application is the following. Here a *quadratic hypersurface* is the affine or projective algebraic set defined by one polynomial of total degree 2.

Theorem 1.10 *There exists an efficient deterministic algorithm that, given a finite field* $\mathbb{F}$ *of odd characteristic and a quadratic hypersurface* V *over* $\mathbb{F}$ *with* $\dim V > 0$*, computes a rational point on* V*.*

This Theorem is also a crucial step in the proof of the next result, which will be given in detail in a forthcoming publication.

Theorem 1.11 *There exists an efficient deterministic algorithm that, given a finite field* $\mathbb{F}$ *and a Weierstrass equation for an elliptic curve* E *over* $\mathbb{F}$*, computes a rational point on* E*, other than the point at infinity, or shows that no such point exists.*

Also, all algorithms given in the thesis can serve as stepping-stones for algorithms to solve the corresponding problems over p-adic rings and fields.

1.5 Conventions and definitions

Finite fields. We will use the notation $\mathbb{F}$, and sometimes also $\mathbb{E}$, for finite fields, and q for the cardinality of either $\mathbb{E}$ or $\mathbb{F}$. As finite fields are determined up to isomorphism by their cardinality, we may also write $\mathbb{F}_q$, for some prime power $q > 1$, to denote a finite field of q elements, or $\mathbb{F}_p$ for a prime p to denote a prime field.

One of the basic algorithmic questions to be answered is how to represent finite fields and their elements in a computer. There are several reasonable possibilities:

(i) we could view a finite field as a vector space over the prime field equipped with a multiplication table with respect to some basis of the space;

(ii) we could view it as a quotient ring of the ring of polynomials in one variable over the prime field by the ideal generated by an irreducible polynomial;

(iii) as the multiplicative group of a finite field is cyclic, we could even just give a generator of the multiplicative group and represent elements by powers of the generator. This is only practical if the field is small, and addition in this representation usually requires a stored table of some sort.

In [35], Lenstra discusses several of these representations and shows how to go from one to the other efficiently, if possible. In computational practice, usually the second possibility is chosen, in combination with the third if the field is small.

In this thesis, we use the second representation of the list just given. The elements of $\mathbb{F}$ are hence represented as polynomials over the prime field whose degree is less than that of the defining polynomial. It follows that the field operations in $\mathbb{F}$ can be performed by means of efficient deterministic algorithms.

More specifically, addition and multiplication in a finite field of q elements can be done using $\tilde{O}(\log q)$ bit operations, by Theorems 8.23 and 8.24 in [22].

Forms. We will need the following definitions on forms. Here, and in the sequel, we denote polynomial variables by capital letters, and values taken on by those variables by letters in lower case.

A *form* of degree n in v variables over a field F is a homogeneous polynomial $f \in F[X_1, \dots, X_v]$ whose total degree equals n.

Any form f of degree other than zero, being homogeneous, possesses the *trivial zero* $(0, 0, \dots, 0)$. A form f in v variables is said to *represent* a given element $b \in F$ if there exist $x_1, \dots, x_v \in F$, not all zero, such that $f(x_1, \dots, x_v) = b$. Thus, a form must possess a *nontrivial zero* in order to represent 0. A form that represents zero is also called *isotropic*, and a form that does not *anisotropic*; a form over F that represents all nonzero elements in F is called *universal.*

Note that if a form f of degree n represents a nonzero element b, then it also represents all elements of the form bx^n with $x \in \mathbb{F}^*$. In this case also the coset of b modulo $\mathbb{F}^{*n}$ is said to be represented by the form f.

A *diagonal form* can be written as $\displaystyle\sum_{i=1}^{v} a_i X_i^n$, where $a_i \in F^*$ for $i = 1, \dots, v$ and $v \in \mathbb{Z}_{\geq 0}$. In this thesis, diagonal forms are assumed not to have zero coefficients.

Complexity. We will present many asymptotic complexity estimates for algorithms. For these, we assume that our algorithms are executed by a processor that performs *bit operations* and is equipped with an unlimited *random access memory.* The estimates will be presented using either “big-Oh” or “soft-Oh” notation, which are defined as follows. We take the definition of $\tilde{O}$ from [22, Section 25.7]; the symbol $\mathbb{N}$ denotes the set of nonnegative integers.

Definition 1.12 Let $f, g : \mathbb{N} \to \mathbb{R}$ be functions such that there exists $N \in \mathbb{N}$ such that $f(n) > 0$ and $g(n) > 0$ for all $n \geq N$. Then f is $O(g)$ if there exist $N' \in \mathbb{N}$ and a positive $C \in \mathbb{R}$, such that, for all $n \geq N'$,

$$f(n) \leq Cg(n). \tag{1.13}$$

Furthermore, f is $\tilde{O}(g)$ if there exist $N', c \in \mathbb{N}$ such that, for all $n \geq N'$,

$$f(n) \leq g(n) \left(\log(3 + g(n))\right)^c . \tag{1.14}$$

Here the constant 3 is added to $g(n)$ to make sure that the logarithmic factor is eventually bounded away from 1. This entails that $Cg(n)$ is $\tilde{O}(g(n))$ for all positive $C \in \mathbb{R}$ and all eventually positive functions $g : \mathbb{N} \rightarrow \mathbb{R}$; in other words, that f is $\tilde{O}(g)$ whenever f is $O(g)$. We note that we also use the "big-Oh" and "soft-Oh" notations for functions of two or more variables; here the properties (1.13) and (1.14) are required to apply for all values of the variables outside some bounded domain.

We will assume the use of asymptotically fast algorithms for integer and polynomial arithmetic throughout this text. As shown in detail in, for example, [22], the adoption of the Schönhage-Strassen algorithm for fast multiplication [22, Section 8.1] leads to essentially linear time bounds on many basic arithmetic algorithms, such as division with remainder and the extended Euclidean algorithm — see also Sections 9.1 and 11.1 of [22].

Many complexity estimates in this thesis are given in terms of *field operations*, i.e., the elementary operations of addition, multiplication, division, and equality testing within some given computable field. Raising an element to a power is not considered to be an elementary field operation.

Chapter 2

Finite field theory

2.1 Introduction

This chapter relates some elements of finite field theory which will be used in the sequel, such as the fact that in dealing with nth powers in a finite field of q elements, we can always replace n by $d = \gcd(n, q-1)$ (while, of course, the *monomials* X^n and X^d over such a field remain quite distinct), and the properties of the subfield of a finite field generated by the nth powers.

Furthermore, we give a new sufficient condition for the solvability of Problem A2; the sufficiency of this condition, and the fact that Problem A1 is always solvable, are proved in a constructive way, which bears close resemblance to our algorithmic solution of these problems in Chapter 6.

Finally, we use Weil's bound to give upper and lower bounds on the *number* of solutions of Problems A1 and A2. We use these bounds for the analysis of a simple probabilistic algorithm that finds solutions to these problems.

For a fuller account of the results treated in this chapter, see [30], [36].

2.2 The subgroup of nth powers

Let $\mathbb{F}$ be a finite field, and let q be its number of elements. Recall that the multiplicative group $\mathbb{F}^*$ is cyclic. Hence for a positive integer n the quotient of $\mathbb{F}^*$ by its subgroup of nth powers has d elements, where d equals the greatest common divisor of n and the group size $q-1$. It follows that the set of dth powers and the set of nth powers in $\mathbb{F}^*$ coincide, and therefore in computations involving powers in $\mathbb{F}$ we might just as well replace n with the often smaller exponent d.

The algorithms presented in the next Chapters are robust with respect to n, in that they remain valid if n does not divide $q-1$. However, as in practice one would first replace n with d before doing any actual computation, we have required in most places that n divide $q-1$. This also facilitates the analysis of the running time.

At the end of execution of an algorithm that is used to compute some nth power in $\mathbb{F}$, one must convert the dth powers in the output back to nth powers. This is done by the following algorithm, which is clearly correct and deterministic, and spends time polynomial in $\log q$ and n.

Algorithm 2.1 (Convert powers.)
Input: a finite field $\mathbb{F}$ of q elements, a positive integer n and an element a of $\mathbb{F}$.
Output: an element b of $\mathbb{F}$ such that $b^n = a^d$, where $d = \gcd(n, q-1)$.

```
1: Compute integers $x$ and $y$, with $0 < x \leq q-1$, such that $xn + y(q-1) = d$,
   where $d = \gcd(n, q-1)$.
2: Compute $a^x$ and output the result.
```

2.3 The subfield of sums of nth powers

Next, we observe that the set K of elements of $\mathbb{F}$ which can be written as a sum of nth powers in $\mathbb{F}$ is a subfield of $\mathbb{F}$, and derive some interesting properties of this subfield. Here, and henceforth, we abuse language to call *an nth power in* $\mathbb{F}$ what should really be called *the nth power of some element in* $\mathbb{F}$. A subfield $\mathbb{E}$ of $\mathbb{F}$ is called a *proper subfield* if we have $\mathbb{E} \neq \mathbb{F}$.

Proposition 2.2 *Let $\mathbb{F}$ be a finite field of q elements and characteristic p, let n be a positive integer and let K be the subset of all sums of nth powers in $\mathbb{F}$. Then:*

(i) *K is a subfield of $\mathbb{F}$.*

(ii) *If $q = p^f$ and $|K| = p^e$, then $(p^f-1)/\gcd(n, p^f-1)$ divides $p^e - 1$ and e is the smallest divisor of f with this property.*

(iii) *If K is a proper subfield of $\mathbb{F}$, then we have $n^2 > q$.*

(iv) *The field K can be generated by adjoining an nth power in $\mathbb{F}$ (that need not be an nth power in K) to its prime field $\mathbb{F}_p$.*

(v) *K is equal to $\mathbb{F}$ if, and only if, $\mathbb{F}$ can be generated as a field by adjunction of an nth power in $\mathbb{F}$ to $\mathbb{F}_p$.*

(vi) *Every nonzero element of K is a sum of nth powers with at most* $\gcd(n,\, q-1)$ *terms.*

Proof. It is clear that K contains 0 and 1 and is closed with respect to addition. As $\mathbb{F}$ has positive characteristic, this also implies closure under subtraction. Furthermore, if we multiply two sums of nth powers into each other term by term, the product will again be such a sum. As to quotients, if b is a nonzero sum of nth powers we can write its multiplicative inverse in the same form by computing b^{n-1} and dividing through by b^n. This proves the first statement.

Write d for the greatest common divisor of n and $p^f - 1$. Clearly, K is the smallest subfield of $\mathbb{F}$ whose multiplicative group contains the set of all nonzero nth powers in $\mathbb{F}$ as a subgroup; this set has $(p^f-1)/d$ elements. This implies the second statement.

Suppose that K is a proper subfield of $\mathbb{F}$. We just saw that K has at least $(q-1)/d$ nonzero elements; on the other hand, in this case we find $|K| \leq \sqrt{q}$. We find the inequalities

$$\frac{q-1}{n} \leq \frac{q-1}{d} \leq |K|-1 \leq \sqrt{q}-1,$$

so that $n \geq \sqrt{q}+1$, which shows the truth of the third statement.

The fourth statement is easy. Let b be a generator of the multiplicative group of $\mathbb{F}$; then $\mathbb{F}_p(b^n)$ contains all nth powers in $\mathbb{F}$, and it is clearly the smallest subfield of $\mathbb{F}$ with this property; therefore, it is equal to K.

As to the fifth statement, if b^n generates $\mathbb{F}$ over its prime field $\mathbb{F}_p$ for some $b \in \mathbb{F}$, then we have $K = \mathbb{F}$, because K contains $\mathbb{F}_p(b^n)$ for all $b \in \mathbb{F}$. The converse follows trivially from the fourth statement.

Finally, write S_s for the set of nonzero elements of $\mathbb{F}$ that are sums of nth powers in $\mathbb{F}$ of at most s terms (for $s = 1, 2, \ldots$). We have

$$\mathbb{F}^{*n} = S_1 \subseteq S_2 \subseteq \cdots \subseteq K^*.$$

There are at most $d-1$ strict inclusions in this chain, because with every new element represented as a sum of s powers, also its whole coset modulo $\mathbb{F}^{*n}$ is so represented, and the number of such cosets contained in K^* is at most d. Furthermore, if S_s equals S_{s+1} for some s, it also equals all later terms in the chain; hence this can happen only if S_s is equal to K^*. It follows that $S_d = K^*$. ♦

The results above are due in essence to Tornheim [53]; see [30, Section 4.2] for further results and references. Note that the fact that K is closed under quotients could also be proved more directly, by observing that b^{q-2} is the multiplicative inverse of a nonzero b in K; however, the proof given above works for *any* field of nonzero characteristic.

Below we will give efficient deterministic algorithms for computing a field generator which is an nth power (as in (iv); Algorithm 4.14) and representing nonzero elements as sums of nth powers with at most $\gcd(n,\, q-1)$ terms (as in (v); Algorithm 5.22).

2.4 The existence of solutions

Among the multivariate forms over a finite field, diagonal forms occupy a special place, and among these a special place is taken by the 'sum-of-powers' form $\sum X_i^n$. We will put this in evidence by proving Theorem 2.3 below by a *constructive* method, which is only suited to diagonal forms, and in which the 'sum-of-powers' form plays a special role.

Theorem 2.3 asserts that Problem A1 is always solvable, and gives a sufficient condition for the solvability of Problem A2. We refer to Section 1.5 for definitions.

Theorem 2.3 *Let $\mathbb{F}$ be a finite field of q elements, and let n be a positive integer.*

(i) *Every diagonal form of degree n over $\mathbb{F}$ in $n+1$ variables is isotropic.*

(ii) *Assume every element in $\mathbb{F}$ is a sum of nth powers in $\mathbb{F}$. Then every diagonal form of degree n over $\mathbb{F}$ in n variables is universal.*

The first statement is part of the Chevalley-Warning theorem; the proof given below is due to Dem′yanov (see [20] and [30, Théorème 4.1]), and independently to Kneser for the case $n = 2$ (see [32, Theorem XI.4.4]).

It turns out that Dem′yanov's method can be extended to yield a proof of the second statement. This statement was first proved by Schwarz [44] on the stronger assumption that $d = \gcd(n,\, q-1) < p$, where p is the characteristic of $\mathbb{F}$; however, his (nonconstructive) proof is still valid if we know that the field $\mathbb{F}$ can be generated over its prime field by an nth power. By Proposition 2.2(v), this is equivalent to our assumption.

One may ask whether Theorem 2.3(ii) gives a *necessary* criterion for all diagonal forms of degree n in n variables to be universal. The answer depends on the exact formulation. On the one hand, if the criterion fails, then *eo ipso* the sum-of-powers form $\sum_{i=1}^{n} X_i^n$ is not universal. On the other hand, if for given n and $\mathbb{F}$ the elements $a_1, \ldots, a_n \in \mathbb{F}^*$ cover the residue classes modulo $\mathbb{F}^{*n}$, then $\sum_{i=1}^{n} a_i X_i^n$ is universal by construction; it follows that universal forms exist even if the criterion fails.

Proposition 2.4 *Let $\mathbb{F}$ be a finite field of q elements, let d be a nonnegative integer and let $a_0, \ldots, a_d$ be a sequence of nonzero elements of $\mathbb{F}$. For $k = -1, 0, \ldots, d$, write $f_k = \sum_{i=0}^{k} a_i X_i^n$, and let S_k be the set of all nonzero elements of $\mathbb{F}$ that are represented by f_k (thus, f_{-1} is the zero form, and S_{-1} is empty).*

(i) *For $k = 0, \ldots, d$, if f_k is not isotropic, then $S_{k-1} \subsetneq S_k$.*

(ii) *Suppose every element of $\mathbb{F}$ is a sum of nth powers in $\mathbb{F}$. Then for $k = 0, \ldots, d$, if $S_{k-1} \neq \mathbb{F}^*$, then $S_{k-1} \subsetneq S_k$.*

Example. An example where we have $S_k = S_{k-1}$ because the form f_k becomes isotropic, is given by the form $X^3 + Y^3$ over $\mathbb{F}_4$, with $k = 2$; every sum of two third powers in $\mathbb{F}_4$ is itself a third power, hence X^3 and $X^3 + Y^3$ represent the same number of nonzero elements, while a nontrivial zero of $X^3 + Y^3$ is given by $X = Y = 1$. In fact, all third powers in $\mathbb{F}_4$ are contained in the prime field $\mathbb{F}_2$; it follows that the elements of $\mathbb{F}_4 \backslash \mathbb{F}_2$ are not sums of third powers in $\mathbb{F}_4$.

Proof. In all cases, we obviously have $\emptyset = S_{-1} \subseteq S_0 \subseteq S_1 \subseteq \ldots \subseteq S_d \subseteq \mathbb{F}^*$; we must show that some of these inequalities are *strict*.

Let $0 \leq k \leq d$; we prove (i). If f_{k-1} is isotropic, then so is f_k; therefore assume that f_{k-1} does not represent zero. Let $Y_1, Y_2, \ldots$ be new variables, and let h be the smallest positive integer for which $a_k(Y_1^n + \ldots + Y_h^n)$ represents at least one element b that is not represented by f_{k-1}. Such an h exists, and we have $h \leq p = \operatorname{char} \mathbb{F}$, because we can represent $b = 0$ nontrivially as a_k times a sum of p terms 1^n. By the minimality of h, it follows that, for some $y_1, \ldots, y_h, x_0, \ldots, x_{k-1} \in \mathbb{F}$, we have

$$b - a_k y_h^n = a_k(y_1^n + \ldots + y_{h-1}^n) = f_{k-1}(x_0, \ldots, x_{k-1}),$$

and hence b is represented by $f_{k-1} + a_k X_k^n$, which is f_k. If $b = 0$, this shows that f_k is isotropic, and if not, we find that $S_k \supsetneq S_{k-1}$, as desired.

Suppose now that every element of $\mathbb{F}$ is a sum of nth powers. Let $1 \leq k \leq d$; we prove (ii), in analogy to the proof of (i). This time, let h be the smallest positive integer for which $a_k(Y_1^n + \ldots + Y_h^n)$ represents at least one *nonzero* element b that is not represented by f_{k-1}. Such an h exists, because $S_{k-1} \neq \mathbb{F}^*$, and because all elements of $\mathbb{F}^*$ are sums of nth powers. Now the same argument as above shows that S_k is strictly larger than S_{k-1}. ♦

Proof of Theorem 2.3. Proposition 2.4(i) implies that a diagonal form of degree n in $n+1$ variables is isotropic, since otherwise it would have to represent $n+1$ classes modulo $\mathbb{F}^{*n}$, of which at most n exist. This proves the first part.

Suppose every element of $\mathbb{F}$ is a sum of nth powers. In this case, 2.4(ii) entails that every diagonal form of degree n in n variables represents, if not all of $\mathbb{F}^*$, then at least n classes of $\mathbb{F}^*$ modulo nth powers. But of such classes there are at most n. This proves the second part. ♦

Generalisations. It should be noted that the proof just given does not use the finiteness of $\mathbb{F}$ at all. In fact, for any field F where -1 is a sum of nth powers, the same proof shows that a diagonal form of degree n over F with more than $|F^*/F^{*n}|$ variables has a nontrivial zero. This is the statement originally proved by Dem′yanov [20].

A field K is said to *have the property C_d* or to *be a C_d-field* if any form over K of degree n in more than n^d variables has a nontrivial zero over K. Now for any odd prime p and any positive integer n, the subgroup of nth powers has index at most n^2 in $\mathbb{Q}_p^*$ (even if p divides n), where $\mathbb{Q}_p$ denotes the field of p-adic numbers. Therefore, by Dem′yanov's result, the field $\mathbb{Q}_p$ has the property C_2 when restricted to *diagonal forms*, whenever p is odd; by a more subtle argument, Dem′yanov proves the same for $\mathbb{Q}_2$ (in [20]). However, Artin's conjecture that the C_2-property holds in general for these fields has proved false. For a discussion of these interesting properties of the p-adic fields, including the first counter-examples to Artin's conjecture, see [25, Chapter 7]; more recently, counter-examples to Artin's conjecture have been constructed for infinitely many degrees (see [60] for a discussion).

2.5 The Weil bound

In 1949, André Weil published bounds on the number of solutions of equations of the form

$$a_0 x_0^{n_0} + a_1 x_1^{n_1} + \ldots + a_v x_v^{n_v} = 0$$

over a finite field (see [56]). The following theorem gives his results, applied to the case of homogeneous equations, where we have $n_0 = n_1 = \ldots = n_v = n$. We use these bounds in the analysis of the probabilistic solution method for such equations that is given the next Section.

Theorem 2.5 *[56, p. 502]*
Let $\mathbb{F}$ be a finite field of q elements, and n and v positive integers. For any elements $a_0, \ldots, a_v \in \mathbb{F}^*$, the number N of $(x_0, \ldots, x_v) \in \mathbb{F}^{v+1}$ such that $\sum_{i=0}^{v} a_i x_i^n = 0$ satisfies

$$|N - q^v| \le (n-1)^v q^{(v+1)/2}; \tag{2.6}$$

for any $b \in \mathbb{F}^*$, the number N_b of $(x_0, \ldots, x_v) \in \mathbb{F}^{v+1}$ such that $\sum_{i=0}^{v} a_i x_i^n = b$ satisfies

$$|N_b - q^v| \le (n-1)^{v+1} q^{v/2}. \tag{2.7}$$

This result, also found as Theorems 6.36 and 6.37 in [36], is part of the so-called Weil conjectures on varieties over finite fields, which Weil formulated in the cited paper [56] and proved for the case of diagonal varieties.

From this result we will derive some interesting conditions on q, n, and v which ensure solvability or the presence of "many" solutions. In this context, Theorem 2.5 gives us nothing if $q < (n-1)^2$: all lower bounds become negative in that case.

Corollary 2.8 *Let $\mathbb{F}$ be a finite field of q elements and n an integer with $n \ge 2$.*

(i) *If $q > (n-1)^4$, then every diagonal form of degree n in 2 variables over $\mathbb{F}$ is universal.*

(ii) *If $q > (n-1)^{2+\frac{2}{n-1}}$, then every diagonal form of degree n in n variables over $\mathbb{F}$ is universal.*

(iii) *Let δ be real with $0 < \delta < 1$. If $q > (n-1)^2$ and v is an integer satisfying $v \ge 2\frac{\log(n-1)-\log(1-\delta)}{\log q - 2\log(n-1)}$, then every diagonal form of degree n in $v+1$ variables over $\mathbb{F}$ represents every nonzero element of $\mathbb{F}$ in at least δq^v different ways.*

(iv) *If $q \ge 4(n-1)^2$, then every diagonal form of degree n in n variables over $\mathbb{F}$ represents every nonzero element of $\mathbb{F}$ in at least $\frac{1}{2}q^{n-1}$ different ways.*

Of course, if for some v every diagonal form in v variables is universal, then every diagonal form in $v+1$ variables is isotropic. The converse is false, as shown, for example, by the case of forms over $\mathbb{F}_7$: all ternary cubic diagonal forms over $\mathbb{F}_7$ are isotropic, whereas the binary form $X^3 + Y^3$ is not universal, as it does not represent ± 3.

Proof. Let f be an arbitrary diagonal form of degree n in $v+1$ variables. If we have

$$q > (n-1)^{2+\frac{2}{v}}, \tag{2.9}$$

then by (2.7) the number of representations of any $b \in \mathbb{F}^*$ by the form f is positive, hence the form is universal. We obtain the first two statements by substituting $v = 1$ and $v = n-1$.

Next, we want a good lower bound on *how many* representations of a given $b \in \mathbb{F}$ by f exist. Let δ satisfy $0 < \delta < 1$. If we have

$$(1-\delta)q^v \ge (n-1)^{v+1} q^{v/2}, \tag{2.10}$$

then by (2.7), we find that there exist at least δq^v distinct representations of any nonzero $b \in \mathbb{F}$ by f.

To obtain the third statement, we solve (2.10) for v, by taking logarithms: we find

$$v\left(\frac{\log q}{2} - \log(n-1)\right) \geq \log(n-1) - \log(1-\delta).$$

Now we divide out the coefficient of v, which is positive by assumption.

For the fourth statement, take $\delta = \frac{1}{2}$ and $v = n-1$. Then (2.10) is equivalent to

$$q \geq 4^{\frac{1}{n-1}}\,(n-1)^{2+\frac{2}{n-1}} = g(n-1)\cdot(n-1)^2,$$

where $g : \mathbb{R} \to \mathbb{R}$ is defined by $g(x) = (4x^2)^{\frac{1}{x}}$. We have $g(1) = g(2) = 4$. Furthermore, g is decreasing on $[2, \infty)$ as its derivative is negative for $x > \mathrm{e}/2 \approx 1.4$, so we find $g(n-1) \leq 4$ for all *integers* $n \geq 2$. Therefore, the assumption $q \geq 4(n-1)^2$ implies (2.10) if $n \geq 2$, and it follows that every nonzero $b \in \mathbb{F}$ has at least $\frac{1}{2}q^{n-1}$ representations by f under this assumption. ♦

This Corollary shows that with diagonal forms, the Chevalley-Warning theorem (Theorem 2.3(i)) is far from being sharp with respect to the number of variables, when q is large compared to the degree n. Part (i) even shows that if $q > (n-1)^4$, every equation of the form $aX^n + bY^n = c$, with $abc \neq 0$, is solvable. More results for the case where all coefficients of the form are equal are given in [36] *s.v.* "Waring's problem for finite fields", and in [59].

As regards general forms, the Chevalley-Warning theorem *is* sharp: the norm form of any basis of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$ is an anisotropic form of degree n in n variables over $\mathbb{F}_q$.

2.6 The probabilistic approach

Before we lay out our deterministic method for solving our Problems A1 and A2, we give a straightforward *probabilistic* method for finding a solution to

$$\sum_{i=1}^{n} a_i x_i^n = b \tag{2.11}$$

over a finite field $\mathbb{F}$ of q elements, where the a_i and b are nonzero elements of $\mathbb{F}$, and where n divides $q-1$.

The idea is the following (see also [23]): let $x_1, \ldots, x_{n-1}$ be random elements of $\mathbb{F}$, test whether

$$\left(b - \sum_{i=1}^{n-1} a_i x_i^n\right)/a_n$$

is an nth power in $\mathbb{F}$, and if it is, take its nth root by means of a probabilistic root taking method.

For this to work, however, we must be sure that there are enough solutions, otherwise we are not likely to find one by guessing. Now to every $(n-1)$-tuple $(x_1, \ldots, x_{n-1})$ correspond either zero, one, or n solutions to (2.11). A lower bound for the number of "lucky" elements of $\mathbb{F}^{n-1}$ is thus obtained by dividing the number of solutions to (2.11) by n.

By Corollary 2.8, (iv), we find that if $q \geq 4(n-1)^2$, there are at least $q^{n-1}/2$ representations of b of the form (2.11). Thus if $q \geq 4(n-1)^2$, we may expect that at least one in every $2n$ elements of $\mathbb{F}^{n-1}$ will give rise to one or more solutions of (2.11).

An estimate for the expected running time of this algorithm can be given as follows: every execution needs n exponentiations with exponent n (which take $O(\log n)$ multiplications each), a test whether the result is an nth power (which takes $O(\log q)$ multiplications), and finally, if it is indeed an nth power, a call to a root-taking algorithm. For this one could use the Tonelli-Shanks algorithm, suitably generalised to account for general exponents n, which uses $\tilde{O}(n + (\log q)^2)$ multiplications (see Section 3.2 for details); however, other methods are faster. More precisely, by [22, Corollary 14.16] root taking can be done using an expected number of $\tilde{O}(n \log q)$ field operations. The expected number of iterations of the first two steps is at most $2n$; this yields an expected total of

$$O(n^2 \log n + n \log q)$$

multiplications in $\mathbb{F}$ for the whole algorithm. We assume here that the generation of a random element of $\mathbb{F}$ is about as complex as a multiplication in $\mathbb{F}$, which takes $\tilde{O}(\log q)$ bit operations by Theorems 8.23 and 8.24 in [22].

In short, we have proved the following statement.

Proposition 2.12 *There exists a probabilistic algorithm that, given a finite field $\mathbb{F}$ with q elements, a positive integer n satisfying $4(n-1)^2 \leq q$, and nonzero elements $a_1, \ldots, a_n$ and $b \in \mathbb{F}$, returns a solution to* (2.11)*, using an expected number of $\tilde{O}(n^2(\log q) + n(\log q)^2)$ bit operations.*

Remark. The running time bound of Proposition 2.12 is given in terms of bit operations to facilitate comparison with the results of Chapters 5 and 6.

The probabilistic approach just sketched does not work if q is too small; in particular, if $q < (n-1)^2$, there may exist just one solution to (2.11), and any probabilistic method is very unlikely to find it.

A comparison between the performance of the probabilistic algorithm given above and the deterministic solver developed in the subsequent chapters will be given in Section 7.2.

Chapter 3

Selective root extraction

3.1 Introduction and results

In this chapter we prove the following statement.

Theorem 3.1 *There exists a deterministic algorithm that, given a finite field $\mathbb{F}$ with q elements, a positive integer n, and $n+1$ elements $a_0, \ldots, a_n$ of $\mathbb{F}^*$, determines integers i and j and an element $b \in \mathbb{F}^*$ such that $0 \le i < j \le n$ and*

$$a_i/a_j = b^n,$$

in time polynomial in n and $\log q$.

We will present an explicit algorithm satisfying the conditions of the Theorem in Section 4 below (Algorithm 3.12); it is an essential subroutine in many of the other algorithms presented in this thesis.

The Theorem says that, given $n+1$ elements in $\mathbb{F}^*$, we can compute an nth root of the quotient of two among them. The choice of which two to take is up to the algorithm. Of course, the quotient of the selected elements must be an nth power in $\mathbb{F}^*$, but the criteria used by the algorithm are more subtle than this.

Although it uses parts of the well-known Tonelli-Shanks root taking algorithm, Algorithm 3.12 is *deterministic.* The property that makes this determinism possible is the fact that the computed root b is *contained in the group generated by* $a_0, \ldots, a_n$, whereas an nth root of one given element a is not in general contained in the group generated by a. Therefore, while the Tonelli-Shanks algorithm needs to construct elements that are outside some given subgroup, Algorithm 3.12 circumvents this need. Note that, if $\mathbb{F}$ is a given finite field, we cannot consider the entire multiplicative group $\mathbb{F}^*$ as given, because it is hard to find a generator for this group, or even a set of generators (see, for example, [5]).

In Section 2, we discuss the Tonelli-Shanks algorithm and bound its complexity, in the case of roots of prime power exponent in an arbitrary finite field. (In the literature, usually only the case of square roots in finite prime fields is considered — see e.g. [16, Section 1.5.1].)

Section 3 gives the deterministic adaptation of the Tonelli-Shanks algorithm that will be used in Algorithm 3.12.

In Section 4, we give the proof of Theorem 3.1, consisting of Algorithm 3.12 and its analysis. We end with some remarks on the performance of Algorithm 3.12, and on generalisations of it to more general abelian groups.

Notation. If a is an element of some group G, we write $\mathrm{ord}(a)$ for its order in G. For a prime ℓ and integers a and $b \neq 0$, we write $v_\ell(a/b)$ for the ℓ-adic valuation of a/b, i.e., the number of factors ℓ in a minus the number of factors ℓ in b.

3.2 The Tonelli-Shanks algorithm

The substance of the Tonelli-Shanks algorithm was already given by Tonelli in 1891 [52] for the purpose of extracting square roots modulo primes of the form $4k+1$. It has subsequently been rediscovered by Shanks [46] and by Adleman, Manders, and Miller [1] in the 1970's, all of whom generalise the method to finding roots of arbitrary exponent, while Shanks also notes that the method can be applied to arbitrary cyclic groups.

The Tonelli-Shanks algorithm has two key ideas, which we present before giving the actual algorithm. The first will be reused in our deterministic methods, while the second gives rise to the probabilistic part of the algorithm and will be replaced.

Computations in the Sylow group. The first key idea is embodied in the following Proposition, which is usually applied with $\ell = 2$ and $f = 1$. It shows how to compute an ℓ^fth root of a from an ℓ^fth root of b, where b is constructed from a in such a way that its order has fewer factors ℓ than the order of a. Applying this construction inductively, we will end up with an element the order of which is prime to ℓ, and taking a root of such an element is easy.

Recall that for a finite abelian group G and a prime ℓ, the *ℓ-Sylow subgroup* of G is the subgroup of all elements of G whose order is a power of ℓ.

Proposition 3.2 *Let $\mathbb{F}$ be a finite field having q elements, ℓ a prime and f a positive integer such that ℓ^f divides $q-1$, and g a generator of the ℓ-Sylow subgroup of $\mathbb{F}^*$. Then for all $a \in \mathbb{F}^*$, either:*

(i) $\mathrm{ord}(a)$ *is not divisible by ℓ, or:*

(ii) *there exist unique integers z and v, with $v \geq 0$ and $z \in \{1, \ldots, \ell-1\}$, such that* $\mathrm{ord}(ag^{-z\ell^v})$ *has fewer factors ℓ than* $\mathrm{ord}(a)$.

If (i) *holds, then a is an ℓ^fth power in $\mathbb{F}^*$. If* (ii) *holds, then a is an ℓ^fth power in $\mathbb{F}^*$ if and only if $v \geq f$.*

Proof. Write $q-1 = \ell^r \cdot u$, where u is in $\mathbb{Z}$ and ℓ does not divide u, and let a be in $\mathbb{F}^*$. Then $\operatorname{ord}(a^u)$ is a power of ℓ.

We have $a^u = 1$ if and only if the order of a has no factors ℓ; this gives the first case. Here a is an element of a cyclic group of order u, on which raising to the ℓ^fth power is a group isomorphism, so evidently a has an ℓ^fth root in this group.

Assume we have $a^u \neq 1$. We choose v such that a^u and g^{ℓ^v} have the same order. Let $\chi : \mathbb{F}^* \to \mathbb{F}^*$ be the map that raises its argument to the power $(q-1)/\ell^{v+1}$. The image of χ has order ℓ^{v+1} and is generated by $\chi(g)$. It follows that for any integer z prime to ℓ, both $\chi(a)$ and $\chi(g^{z\ell^v})$ are primitive ℓth roots of unity. Therefore, if we choose z such that these roots of unity are equal, we see that

$$\chi(ag^{-z\ell^v}) = 1,$$

hence the quotient of a and $g^{z\ell^v}$ has fewer factors ℓ in its order than a.

Let us prove the final statement; recall that $\mathbb{F}^*$ is cyclic and that ℓ divides $\operatorname{ord}(a)$. Therefore, a is an ℓ^fth power in $\mathbb{F}^*$ if and only if $v_\ell(\operatorname{ord} a)$ does not exceed $v_\ell(q-1)-f$, which is $v_\ell(\operatorname{ord} g) - f$, because g generates the ℓ-Sylow subgroup. But by our choice of v, we know that $v_\ell(\operatorname{ord} a) = v_\ell(\operatorname{ord} g) - v$; this gives the inequality $v \geq f$. ♦

Remarks. Note that if we have $\ell = 2$, the choice $z = 1$ always works.

Besides being used here, Proposition 3.2 is also at the heart of the well known *index calculus method* for computing discrete logarithms in cyclic groups; cf. [31, p. 102ff.] or [7, Section 7.3] for a discussion.

By applying the Proposition inductively, we construct an algorithm that computes ℓ^fth roots, but needs a generator of the ℓ-Sylow subgroup of $\mathbb{F}^*$ to proceed.

Finding a Sylow group generator. Here we come across the second key idea: if $b \in \mathbb{F}^*$ is *not an ℓth power in* $\mathbb{F}^*$, then b^u is a generator of the Sylow group, where u is defined as in the proof of the Proposition. Furthermore, we easily exhibit a non-ℓth power in $\mathbb{F}^*$ by picking a random element b and testing whether $b^{(q-1)/\ell} = 1$; if b fails the test, then b is not an ℓth power. As a fraction of $(\ell-1)/\ell$ of elements of $\mathbb{F}^*$ are not ℓth powers, we need only test a few elements before we succeed.

The guessing, of course, makes the algorithm probabilistic. However, it is not known how to construct Sylow group generators, or even how to find a non-ℓth power, in a deterministic way. In practice (assuming $\mathbb{F}$ is a prime field $\mathbb{F}_p$), if one just enumerates $1, 2, 3, \ldots$, one quickly encounters a non-power; if one assumes the truth of the Generalised Riemann Hypothesis, this is proven to happen before we reach $2(\log p)^2$; but the best unconditional results only guarantee a non-ℓth power below a bound of $O_\varepsilon(p^{1/(4\sqrt{e})+\varepsilon})$, which is much too large to obtain an efficient algorithm. For non-prime fields, analogous results hold; proofs of the bounds just mentioned can be found in [4] (assuming GRH) and [38] (unconditional).

Algorithm. When we turn the two key ideas into an algorithm, it becomes clear that it is more efficient to apply Proposition 3.2 to a^u, where u is defined as in the proof of the Proposition. That way, the orders of all occurring elements are powers of ℓ, and instead of using the map χ, we already find ℓth roots of unity after raising to the power ℓ^{w-1}, where $w = v_\ell(\operatorname{ord} a)$ (Step 4c). Applying this to the element g^{ℓ^v}, we always obtain the same root of unity $\zeta = g^{\ell^{r-1}}$, where $r = v_\ell(q-1)$. This root is therefore precomputed in Step 3.

In Step 4d, by using $\ell^w - z$ in the exponent rather than $-z$, we avoid taking quotients, which is more expensive than multiplying.

We remark that, as said, the "classical" Tonelli-Shanks algorithm is recovered by taking $\ell = 2$ and $f = 1$.

Algorithm 3.3 (Tonelli-Shanks.)
Input: a finite field $\mathbb{F}$, having q elements, a prime number ℓ and a positive integer f with ℓ^f dividing $q-1$, and an element a of $\mathbb{F}^*$.
Output: an element $b \in \mathbb{F}^*$ such that $b^{\ell^f} = a$, or, if no such b exists, "no solution".

```
1: [Pre-processing] Write $q - 1 = \ell^r \cdot u$, where $\ell$ does not divide $u$. Put $A = a^u$
   and $B = 1$. Compute integers $x$ and $y$ such that $xu + y\ell^f = 1$.
2: [Find generator] Try random elements of $\mathbb{F}^*$ until $h \in \mathbb{F}^*$ is found such that
   $h^{(q-1)/\ell} \neq 1$. Put $g = h^u$, raise $g$ to the power $\ell, \ell^2, \ldots, \ell^{r-1}$, and store these
   values in a table.
3: [Roots of unity] Put $\zeta = g^{\ell^{r-1}}$. Compute $\zeta, \zeta^2, \ldots, \zeta^{\ell-1}$ and store them in a
   sorted table.
4: [Loop] While $A \neq 1$ do:
   a: [Compute order] Compute $w$ such that $\operatorname{ord}(A) = \ell^w$.
   b: [Order too large?] If $w + f > r$, terminate saying there is no solution.
   c: [Compute $z$] Using the table computed in Step 3, find the unique integer
      $z$, with $1 \leq z \leq \ell - 1$, such that $A^{\ell^{w-1}} = \zeta^z$.
   d: [Induction] Replace $A$ by $Ag^{(\ell^w - z)\ell^{r-w}}$ and $B$ by $Bg^{z\ell^{r-w-f}}$, using the
      table computed in Step 2.
5: [Result] Output $B^x a^y$.
```

Lemma 3.4 *Algorithm 3.3, except the search in Step 2, can be run using $O(\ell + \log q + W^2 \log \ell)$ operations in $\mathbb{F}$, where $W = v_\ell(\operatorname{ord} a)$.*

Proof. The powerings and order computations in Steps 1, 2, and 5 each take $O(\log q)$ multiplications. The computation of the powers of ζ in Step 3 takes $\ell - 1$ multiplications.

Define W as $v_\ell(\operatorname{ord} a)$. The number of loop executions in Step 4 is at most equal to W. Steps 4b, 4c, and 4d each take $O(\log \ell)$ operations, using the stored values of g to the power $\ell, \ell^2, \ldots, \ell^{r-1}$. Step 4a takes $O(W \log \ell)$ operations to complete. ♦

Proposition 3.5 *Algorithm 3.3 is correct. It is probabilistic, with expected running time the cost of $O(\ell + (\log q)^2)$ operations in $\mathbb{F}$.*

Proof. The correctness of Algorithm 3.3 follows from Proposition 3.2. The value v used in the Proposition is equal to $r - w$; therefore the algorithm rightly returns "no solution" if, and only if, we have $w + f > r$. Note that we always have $AB^{\ell^f} = a^u$, so that in the end B will be an ℓ^fth root of a^u.

To justify Step 5, note that if $B^{\ell^f} = a^u$, then

$$(B^x a^y)^{\ell^f} = a^{xu+y\ell^f} = a.$$

The running time of the deterministic parts of Algorithm 3.3 is bounded by the cost of $O(\ell + (\log q)^2/\log \ell)$ operations in $\mathbb{F}$, by the Lemma and because $W = O(\log q/\log \ell)$. The cost of trying an element for being a non-power also amounts to $O(\log q)$ multiplications; we expect to try at most 2 elements, as the fraction of non-ℓth powers in $\mathbb{F}^*$ is $(\ell - 1)/\ell$. ♦

Remarks. It is easy to extend Algorithm 3.3 to a version that can take nth roots for an arbitrary integer n. First we reduce to the case where n divides $q - 1$ by means of Algorithm 2.1; then we factorise n and apply Algorithm 3.3 one prime factor after the other.

If Algorithm 3.3 is called often with the same $\mathbb{F}$ and ℓ, then the table of ℓth roots of unity should be precomputed. In another direction, we could reduce the term ℓ in the running time to $\sqrt{\ell}$ by using a baby-step-giant-step technique instead of computing a table.

3.3 A deterministic variant

On examining the role of the Sylow group generator g in Algorithm 3.3, one sees that the property of being a generator is not essential for g to work; all we need is that g generates a *large enough subgroup* of the Sylow group. Therefore I propose the following deterministic variant of Algorithm 3.3, where the element g is specified as part of the input, instead of the algorithm having to find g by itself. It will be used as a subroutine by the Selective Root Algorithm in the next Section. Of course, if the order of g is too low, the algorithm will fail to compute a root.

The following generalisation of Proposition 3.2 gives more details.

Proposition 3.6 *Let $\mathbb{F}$ be a finite field having q elements, ℓ a prime and f a positive integer such that ℓ^f divides $q - 1$, and $g \in \mathbb{F}^*$. Let G be the subgroup of all elements $b \in \mathbb{F}^*$ with $v_\ell(\operatorname{ord} b) \le v_\ell(\operatorname{ord} g)$. Then for all $a \in \mathbb{F}^*$, either:*

(i) $\operatorname{ord}(a)$ *is not divisible by ℓ, or:*

(ii) *there exist unique integers z and v, with $v \ge 0$ and $z \in \{1, \ldots, \ell - 1\}$, such that* $\operatorname{ord}(ag^{-z\ell^v})$ *has fewer factors ℓ than* $\operatorname{ord}(a)$*, or:*

(iii) $\operatorname{ord}(a)$ *has more factors* ℓ *than* $\operatorname{ord}(g)$.

If (i) *holds, then* a *is an* ℓ^f*th power in* G. *If* (ii) *holds, then* a *is an* ℓ^f*th power in* G *if and only if we have* $v \geq f$. *If* (iii) *holds, then* a *is not an* ℓ^f*th power in* G.

Proof. We copy the proof of Proposition 3.2, replacing everywhere $\mathbb{F}^*$ by G and $q-1$ by $|G|$, and g by g^u except when g occurs inside the argument of the character χ. The proof goes through, except that the given element a might not be in G; but this is exactly the third case. ♦

Algorithm 3.7 (Deterministic Tonelli-Shanks.)
Input: a finite field $\mathbb{F}$, having q elements, a prime number ℓ and a positive integer f with ℓ^f dividing $q-1$, and elements g and a of $\mathbb{F}^*$.
Output: either an element $b \in \mathbb{F}^*$ such that $b^{\ell^f} = a$ and $v_\ell(\operatorname{ord} b) \leq v_\ell(\operatorname{ord} g)$, or, if no such b exists, "no solution".

```
1: [Pre-processing] Write $q-1 = \ell^r \cdot u$, where $\ell$ does not divide $u$. Put $A = a^u$ and $B = 1$. Compute integers $x$ and $y$ such that $xu + y\ell^f = 1$.
2: [Roots of unity] Replace $g$ by $g^u$. Compute $s$ such that $\operatorname{ord}(g) = \ell^s$. Put $\zeta = g^{\ell^{s-1}}$. Compute $\zeta, \zeta^2, \ldots, \zeta^{\ell-1}$ and store them in a sorted table.
3: [Loop] While $A \neq 1$ do:
    a: [Compute order] Compute $w$ such that $\operatorname{ord}(A) = \ell^w$.
    b: [Order too large?] If $w + f > s$, return "no solution".
    c: [Compute $z$] Using table lookup, find the unique integer $z$, with $1 \leq z \leq \ell - 1$, such that $A^{\ell^{w-1}} = \zeta^z$.
    d: [Induction] Replace $A$ by $Ag^{(\ell^w - z)\ell^{s-w}}$ and $B$ by $Bg^{z\ell^{s-w-f}}$.
4: [Result] Output $B^x a^y$.
```

Proposition 3.8 *Algorithm 3.7 returns a correct solution whenever one exists, and "no solution" otherwise. It is deterministic and uses* $O(\ell + (\log q)^2)$ *operations in* $\mathbb{F}$.

Lemma 3.9 *Algorithm 3.7 can be run using* $O(\ell + \log q + W^2 \log \ell)$ *operations in* $\mathbb{F}$, *where* $W = v_\ell(\operatorname{ord} a)$.

Proof. Same as for Lemma 3.4. ♦

Proof of Proposition 3.8. As we removed the probabilistic Step 2 from Algorithm 3.3, the result is evidently deterministic. The correctness is clear from Proposition 3.6, where this time we have $v = s - w$. Note that "no solution" here means that either no ℓ^fth root exists in $\mathbb{F}^*$, or the orders of all existing roots have too many factors ℓ (i.e., more than $v_\ell(\operatorname{ord} g)$).

The claimed bound on the running time follows from the Lemma. ♦

Remarks. If Algorithm 3.7 can assume that either one, or both, of $\operatorname{ord}(a)$ and $\operatorname{ord}(g)$ are already powers of ℓ, then it can dispense with raising the element in question to the power u, and also Step 4 becomes trivial if a is not raised to the power u.

3.4 The Selective Root Algorithm

In this section we formulate and analyse the algorithm that proves Theorem 3.1. Thus we are given an exponent n and nonzero elements $a_0, \ldots, a_n$ of some finite field $\mathbb{F}$, and we are to compute an nth root of a_i/a_j for some distinct i and j. We will use the deterministic Tonelli-Shanks variant Algorithm 3.7 to take the required roots; to do this, we must find elements whose order contains enough factors ℓ, for all primes ℓ dividing n.

Example. We first explain the situation by an example: let a, b, and c be nonzero elements of a finite field $\mathbb{F}$. By considering the quadratic characters of a, b, and c, we see that at least one of a/b, a/c, and b/c is a square, let us say a/b; however, this does not enable us to take its square root in a deterministic way, because for this we need an element g with $v_2(\operatorname{ord} g) > v_2(\operatorname{ord}(a/b))$, and it is not known how to generate such an element deterministically.

Therefore, let us consider the orders of the elements themselves. If, for example, $v_2(\operatorname{ord} a) = v_2(\operatorname{ord} b)$, then it follows that $v_2(\operatorname{ord}(a/b))$ will be strictly *smaller* than $v_2(\operatorname{ord} a)$; and we can use a in Algorithm 3.7 to take a square root of a/b. If there are no equalities, we may assume $v_2(\operatorname{ord} a) < v_2(\operatorname{ord} b) < v_2(\operatorname{ord} c)$; it follows that $v_2(\operatorname{ord} c) > v_2(\operatorname{ord}(a/b))$, and we can use c to take a square root of a/b.

General situation. The following Proposition shows that elements of large enough order always exist within the subgroup generated by the a_i; this follows because the wanted nth roots are themselves such elements, and the Proposition claims the existence of nth roots. In fact, it shows that the possibilities for finding suitable elements increase if the number of a_i exceeds some *multiple* mn of the exponent n.

Proposition 3.10 *Let $\mathbb{F}$ be a finite field having q elements, n a positive integer dividing $q-1$, and $a_0, \ldots, a_k \in \mathbb{F}^*$, with $k \geq mn$ for some positive integer m. Let G be the subgroup of $\mathbb{F}^*$ generated by the a_i. Then the a_i can be reordered such that a_i/a_j is an nth power in G for all $i, j \in \{0, \ldots, m\}$.*

Proof. The group G is cyclic and is thus partitioned into at most n cosets modulo the subgroup of nth powers. We are given at least $mn+1$ elements of G. Hence by the pigeon hole principle, there exists at least one coset to which at least $m+1$ of the a_i belong. ♦

The Proposition does not show how to find such elements. Now let ℓ be a prime divisor of n, and write $f = v_\ell(n)$; note that it is possible to compute a generator of the ℓ-Sylow subgroup of G (though *not* of $\mathbb{F}^*$, in general) simply by considering which of the a_i has the most factors ℓ in its order. Given such a generator g, the next task is to identify those i and j such that

$$v_\ell(\operatorname{ord}(a_i/a_j)) \leq \min\{0, v_\ell(\operatorname{ord} g) - f\}, \tag{3.11}$$

because that is the exact criterion for an ℓ^fth root of a_i/a_j to exist in G (cf. Proposition 3.6). If $v_\ell(\operatorname{ord} g)$ is less than f, then the ℓ-Sylow subgroup of G does not contain ℓ^fth powers other than 1, and hence the order of a_i/a_j must be prime to ℓ for a_i/a_j to be an ℓ^fth power in G.

By the Proposition, there is a subset S of $\{0, \ldots, n\}$ with at least $(n/\ell^f) + 1$ elements such that the criterion (3.11) is satisfied for all i and j in S. The method is now applied recursively to those a_i with i in the subset S, to identify two among them whose quotient is an mth power, with $m = n/\ell^f$, and compute an mth root of this quotient.

These considerations yield the following algorithm.

Algorithm 3.12 (Selective root taking.)
Input: a finite field $\mathbb{F}$ having q elements, a positive integer n dividing $q - 1$, and nonzero elements $a_0, \ldots, a_n$ of $\mathbb{F}$.
Output: integers i and j, with $0 \leq i < j \leq n$, and an element $b \in \mathbb{F}$ such that $a_i/a_j = b^n$.

1: [Base case] If $n = 1$, output $i = 0$, $j = 1$, and $b = a_0/a_1$, and terminate.

2: [Factor n] Find a prime divisor ℓ of n. Let u be the largest divisor of $q-1$ which is prime to ℓ; put $f = v_\ell(n)$.

3: [Compute orders] For $i = 0, \ldots, n$, compute $A_i = a_i^u$, and compute w_i such that ℓ^{w_i} is the order of A_i.

4: [Find generator] Select j such that w_j is maximal among the w_i, then put $w = w_j$ and $g = A_j$. [The ℓ-Sylow subgroup of $\langle a_0, \ldots, a_n \rangle$ is generated by g and has order ℓ^w.]

5: [Small group?] If $w \leq f$, then for $i = 0, \ldots, n$, put $B_i = A_i$; otherwise, for $i = 0, \ldots, n$, put $B_i = A_i^{\ell^{w-f}}$.

6: [Find equals] Put $m = n/\ell^f$. Among the elements $a_0, \ldots, a_n$, find $m + 1$ such that the corresponding B_i are all equal.

7: [Recur] Apply Algorithm 3.12 recursively to $\mathbb{F}$, m, and the $m + 1$ elements selected in Step 6; we find integers i and j, with $0 \leq i < j \leq n$, and an element $c \in \mathbb{F}^*$ such that $c^m = a_i/a_j$.

8: [Take root] Using Algorithm 3.7 with arguments $\mathbb{F}$, ℓ, f, g, and c, compute b such that $b^{\ell^f} = c$.

9: [Result] Output i, j, and b.

Proposition 3.13 *Algorithm 3.12 is correct and deterministic, and can be done in* $O(n(\log q) + (\log q)^2)$ *operations in* $\mathbb{F}_q$.

Lemma 3.14 *Algorithm 3.12 takes* $O(n(\log q) + \sum_{\ell \mid n}(V_\ell^2 \log \ell))$ *operations in* $\mathbb{F}_q$ *to complete, where* $V_\ell = \max_{0\leq i,j\leq n} v_\ell(\mathrm{ord}(a_i/a_j))$, *and where* ℓ *runs over the prime divisors of* n.

Proof. The running time of the algorithm is taken up by $O(n \log q)$ multiplications in Steps 1–6, the cost of the recursive call in Step 7, and $O(\ell + \log q + V_\ell^2 \log \ell)$ operations in Step 8; here we apply Lemma 3.9, and write V_ℓ for $\max_{i,j} v_\ell(\mathrm{ord}(a_i/a_j))$. The recursive call is for $m = n/\ell^f \leq n/2$ variables. Summing the cost over all recursion levels, and writing $\omega(n)$ for the number of distinct prime divisors of n, we find

$$O\left(\sum_{i=1}^{\omega(n)} \frac{n}{2^{i-1}}(\log q) + \sum_{\ell \mid n}\left(\ell + \log q + V_\ell^2 \log \ell\right)\right) = O\left(n(\log q) + \sum_{\ell \mid n} V_\ell^2 \log \ell\right)$$

operations in $\mathbb{F}_q$. ♦

Proof of Proposition 3.13. We first prove correctness. If $n = 1$, the Proposition is trivial. Thus, let ℓ be a prime dividing n and put $f = v_\ell(n)$. Let u be as computed in Step 2. Consider the subgroup G of $\mathbb{F}_q^*$ generated by the a_i; let H be the ℓ-Sylow subgroup of G.

With these notations, the elements A_i computed in Step 3 are in H, and any one with maximal order among them generates H.

We claim that the test in Steps 5 and 6 is equivalent to the criterion (3.11). Namely, let i and j be such that a_i/a_j has an ℓ^fth root in G. If $w < f$, the only ℓ^fth power in H is 1, and $\mathrm{ord}(a_i/a_j)$ is prime to ℓ; equivalently, we have $A_i = A_j$. But in this case, we also have $B_i = A_i$ for all i. If $w \geq f$, then we have $v_\ell(\mathrm{ord}(a_i/a_j)) \leq v_\ell(g) - f = w - f$; equivalently, the ℓ^{w-f}th powers of A_i and A_j agree. But B_i and B_j are exactly these powers.

By Proposition 3.10, it follows that the selection procedure in Step 6 will be successful. In particular, the indices i and j selected in Step 7 are such that $c^{mu} = (a_i/a_j)^u$ is an ℓ^fth power in H. Therefore, c^u is an ℓ^fth power in H, as m and ℓ are coprime. This is sufficient for Step 8 to succeed. The correctness of Step 7 is clear.

The given bound for the running time follows from the Lemma because we have $V_\ell = O(\log q / \log \ell)$ for all $\ell \mid n$; recall that n is assumed to divide $q - 1$. ♦

Proof of Theorem 3.1. Algorithm 3.12 claims to perform the task described in the Theorem. By Proposition 3.13, it is correct and deterministic, and finishes in time polynomial in n and $\log |\mathbb{F}|$. ♦

Remarks. It is easy to give an iterative formulation for Algorithm 3.12, which probably also results in better performance; a feature that should be kept is the way in which the number of a_i that need to be examined is at least halved in each recursive level.

The number of field operations performed by Algorithm 3.12 is $O((\log q)^2)$ for a field of cardinality q, and thus the algorithm has essentially cubic bit complexity. We have been unable to obtain an essentially quadratic bound, except in situations where the orders of the arguments a_i are bounded independently from q (see Section 5.5 for an example), or in fields where we have $v_\ell(q-1) = O(\sqrt{\log q / \log \ell}\,)$ for all primes ℓ dividing both n and $q-1$ (cf. [9]).

The complexity would improve if we could replace Algorithm 3.7 by a faster root taking algorithm. Now there do exist essentially quadratic probabilistic algorithms for root taking, which are mostly guises of Berlekamp's polynomial factorisation algorithm (see [4] or [22, Section 14.5], for example); but these do not seem to suit the deterministic application in Algorithm 3.12.

Generalisations. The algorithms presented in this Chapter are valid in principle for arbitrary finite cyclic groups, as was already noted by Shanks [46]. However, to obtain efficient algorithms, one has to assume that the group order is known (though not necessarily in factorised form).

In not necessarily cyclic abelian groups, such as the groups $(\mathbb{Z}/m\mathbb{Z})^*$ for positive integers m, one runs into the problem that the order of an element does not determine the subgroups of which it is a member. However, one can still use Algorithm 3.7 to decide, given a positive integer n and group elements a and b, whether the group generated by a contains an nth root of b, and, if so, to compute such a root.

Furthermore, if we can compute the cardinality of the quotient G/G^n for an arbitrary finite abelian group G, then we can adapt Algorithm 3.12 to work also for G. Namely, if this cardinality is m, then it is clear that among every $m+1$ elements $g_0, \ldots, g_m$ of G there must be two whose quotient is an nth power in G. It is then also true that the quotient H/H^n, where H is the subgroup of G generated by the g_i, cannot have more than m elements, so that an nth root of such a quotient g_i/g_j will be contained in the group H.

Chapter 4

Field generators in multiplicative subgroups

4.1 Introduction and results

Let $\mathbb{F}$ be a finite field and n a positive integer. Proposition 2.2 tells us that if every element of $\mathbb{F}$ is a sum of nth powers in $\mathbb{F}$, then there exists some α in $\mathbb{F}$ such that α^n generates $\mathbb{F}$ over its prime field. This leads us to the question whether such an element α can be computed efficiently (and deterministically).

In this chapter we show that this is indeed possible. We use the following auxiliary result, which may be considered as a multiplicative version of the primitive element theorem for separable field extensions (e.g., Theorem V.4.6 in [33]).

Theorem 4.1 *There exists a deterministic algorithm that, given finite fields $\mathbb{E}$ and $\mathbb{F}$, with $\mathbb{E} \subseteq \mathbb{F}$, and nonzero elements $\alpha_1, \ldots, \alpha_t$ of $\mathbb{F}$, computes integers $x_1, \ldots, x_t$ such that*

$$\alpha_1^{x_1} \cdots \alpha_t^{x_t} \text{ generates the field } \mathbb{E}(\alpha_1, \ldots, \alpha_t) \text{ over } \mathbb{E},$$

and uses time polynomial in t and $\log |\mathbb{F}|$.

Section 3 is devoted to the presentation and analysis of an algorithm satisfying the conditions of this Theorem.

In Section 4, we prove the main result of this chapter.

Theorem 4.2 *There exists a deterministic algorithm that, given a finite field $\mathbb{F}$ of q elements and characteristic p, and a positive integer n, computes $\alpha \in \mathbb{F}$ such that $\mathbb{F}_p(\alpha^n) = K$, in time polynomial in n and $\log q$, where K is the subfield of sums of nth powers in $\mathbb{F}$.*

The proof is by an inductive application of Theorem 4.1. Note that we have $K = \mathbb{F}$ whenever $n^2 < q$; see Proposition 2.2. We end the chapter with some generalisations.

Conventions. In this Chapter, $\mathbb{F}/\mathbb{E}$ will denote an extension of finite fields of degree e, and we will use q to denote the cardinality of $\mathbb{E}$. If $\mathbb{F}/\mathbb{E}$ is part of the input of an algorithm, we assume that $\mathbb{F}$ is given as $\mathbb{E}[X]/(f)$, where $f \in \mathbb{E}[X]$ is an irreducible polynomial, and we write β for the formal field generator $X \pmod f$.

4.2 Computing degrees

We will have many occasions to compute the *degree* of an element α of a given finite field $\mathbb{F}$ over a subfield $\mathbb{E}$. The fastest way that I know for doing this is by using the *iterated Frobenius* algorithm, a technique based on fast multipoint evaluation of polynomials, that computes the value of repeated application of the Frobenius map in a fast way comparable to repeated squaring. For details, I refer to [22, Section 14.8].

Algorithm 4.3 (Iterated Frobenius; Algorithm 14.26 in [22].)
Input: a finite field $\mathbb{F}$ of q elements, a squarefree polynomial f over $\mathbb{F}$ of degree n, a positive integer d with $d \leq n$, the element ξ^q where $\xi \in \mathbb{F}[X]/(f)$ is the class of X, and an element $\alpha \in \mathbb{F}[X]/(f)$.
Output: the elements $\alpha, \alpha^q, \ldots, \alpha^{q^d} \in \mathbb{F}[X]/(f)$.

Lemma 4.4 *Algorithm 4.3 is correct and deterministic. It finishes using $\tilde{O}(n^2)$ operations in $\mathbb{F}$.*

Proof. Theorem 14.27 in [22]. ♦

Algorithm 4.5 (Compute the degree of a finite field element.)
Input: finite fields $\mathbb{E} \subseteq \mathbb{F}$, where $|\mathbb{E}| = q$ and $[\mathbb{F} : \mathbb{E}] = e$, the element β^q where β is the given generator for $\mathbb{F}$ over $\mathbb{E}$, and an element α in $\mathbb{F}$.
Output: the degree of α over $\mathbb{E}$.

```
1: Let $f$ be the minimal polynomial of $\beta$ over $\mathbb{E}$. Call Algorithm 4.3, with arguments $\mathbb{E}$, $f$, $e-1$, $\beta^q$, and $\alpha$, to find the elements $\alpha, \alpha^q, \ldots, \alpha^{q^{e-1}}$.
2: Let $d$ be the smallest positive integer such that $\alpha = \alpha^{q^d}$. Output $d$ and terminate.
```

Proposition 4.6 *Algorithm 4.5 is correct and deterministic. Its running time is bounded by the cost of $\tilde{O}(e^2)$ operations in $\mathbb{E}$.*

Proof. An element α has degree d over $\mathbb{F}$ if and only if d is the smallest positive integer for which $\alpha^{q^d} = \alpha$. The Proposition follows by the Lemma (with a different notation!). ♦

4.3 The compositum algorithm

Our compositum algorithm is based on the following key observation (cf. [6, Lemma 6.2]), in which ϕ denotes Euler's totient function, and which we give for arbitrary fields.

Lemma 4.7 *Let L/K be a finite cyclic extension of fields. Then in every basis for L as a K-vector space there are at least $\phi([L:K])$ elements b with the property that $L = K(b)$, and this inequality is best possible.*

Proof. In this proof, an *intermediate field* means any field M with $K \subseteq M \subseteq L$.

We start by constructing a basis for L over K with the property that this basis contains, as subsets, bases for *all* intermediate fields — for brevity, we will call such a basis *complete.*

Assume first that the degree $[L:K]$ is a prime power ℓ^e, and apply induction on e. If M is the unique intermediate field with $[L:M] = \ell$, then by induction there is a complete basis $b_1, \dots, b_{\ell^{e-1}}$ for M over K; but now, if $c_1, \dots, c_\ell$ is a basis for L over M that has $c_1 \in K$, then the set of all products $b_i c_j$ (with $1 \leq i \leq \ell^{e-1}$ and $1 \leq j \leq \ell$) is a complete basis for L.

Second, if the degree $[L:K]$ factors as $\prod_{i=1}^{t} \ell_i^{e_i}$ with $t > 1$ and the ℓ_i distinct primes, we write L as a tensor product

$$M_1 \otimes_K \cdots \otimes_K M_t$$

where M_i denotes the unique intermediate field of degree $\ell_i^{e_i}$ over K. By the first step, we can find complete bases for the M_i; but then the tensor product of these bases is a complete basis for L.

Next, we prove that a complete basis for L/K contains exactly $\phi([L:K])$ field generators, using the same two steps.

In the case where L is of prime power degree and M such that $[L:M] = \ell$, every element that is not in M generates L, and hence a complete basis for L contains exactly $\ell^e - \ell^{e-1} = \phi([L:K])$ field generators.

In the general case, notice that elements of the constructed complete basis for L are tensors of t components, where the ith component belongs to a complete basis for M_i. Such a tensor generates L over K if and only if for $i = 1, \dots, t$, the ith component generates M_i over K. Now we just proved that for each i, a complete basis for M_i contains $\phi([M_i:K])$ field generators; therefore by multiplicativity, exactly $\phi([L:K])$ elements in the complete basis for L each individually generate L over K.

Thus, in a complete basis, the $[L:K] - \phi([L:K])$ basis elements that are not field generators contain among themselves bases for *all* intermediate fields, excepting L itself; this proves that the space V that is additively generated by all these fields has dimension $[L:K] - \phi([L:K])$ over K.

Hence in an arbitrary basis for L over K, at most $[L:K] - \phi([L:K])$ elements can lie in proper subfields of L, and hence in V; all the others must be field generators. ♦

Remark. Actually, as $\liminf_{n\to\infty} \frac{\phi(n)\log\log n}{n}$ exists and is positive (by Theorem 328 in [28]), it is to be expected that at least one in every $c \log\log([L:K])$ elements of a basis for L over K is a field generator, where c is a positive absolute constant.

Algorithm. Using the Lemma, it is easy to formulate an algorithm that satisfies the conditions of Theorem 4.1 above.

Let $\mathbb{E}$ be a finite field with q elements, let t be a positive integer, and for $i = 1, \ldots, t$, let α_i be algebraic over $\mathbb{E}$ of degree e_i. We assume that all α_i are contained in some finite overfield $\mathbb{F}$ of $\mathbb{E}$, and hence the composite field $M = \mathbb{E}(\alpha_1, \ldots, \alpha_t)$ is well defined. Its degree is equal to $\operatorname{lcm}(e_1, \ldots, e_t)$; we denote this degree by d. The notation M for the composite field, and d for its degree, is kept throughout this Section and the next.

Now given $\alpha_1, \ldots, \alpha_t$, compute a basis for $M = \mathbb{E}(\alpha_1, \ldots, \alpha_t)$ over $\mathbb{E}$; then test all the basis elements for being a field generator. We must find at least one generator by virtue of Lemma 4.7.

The form of a basis for M, and the test for being a field generator that we apply follow from the next two Lemmata.

Lemma 4.8 *Write $M_i = \mathbb{E}(\alpha_1, \ldots, \alpha_i)$ for $1 \le i \le t$, and $M_0 = \mathbb{E}$. The degree f_i of M_i over M_{i-1} satisfies, for $i \ge 1$,*

$$f_i = \frac{[\mathbb{E}(\alpha_i) : \mathbb{E}]}{\gcd(\,[M_{i-1} : \mathbb{E}], [\mathbb{E}(\alpha_i) : \mathbb{E}]\,)}.$$

A basis for $M = M_t$ over $\mathbb{E}$ is given by $\left\{ \prod_{i=1}^{t} \alpha_i^{x_i} \mid 0 \le x_i \le f_i - 1 \right\}$.

Proof. We have $M_i = M_{i-1}(\alpha_i)$, so the powers of α_i give a basis for M_i over M_{i-1}. The claim for the relative extension degree f_i follows by the fact that finite extensions of finite fields are cyclic, and therefore subextensions are uniquely determined by their degrees.

A basis for M_i over $\mathbb{E}$ is thus given by $\{\beta\alpha_i^x \mid \beta \in \mathcal{B},\, 0 \le x \le f_i - 1\}$, if $\mathcal{B}$ is a basis for M_{i-1} over $\mathbb{E}$. The statement now follows by induction. ♦

Lemma 4.9 *Let $\alpha \in M$. Then we have $M = \mathbb{E}(\alpha)$ if and only if*

$$\alpha^{q^{d/\ell}} \neq \alpha$$

for all prime divisors ℓ of d, where $d = [M : \mathbb{E}]$.

Proof. An element of M is a field generator over $\mathbb{E}$ if and only if it lies outside all *maximal* subfields of M that contain $\mathbb{E}$. These maximal subfields are in one-to-one correspondence to the prime divisors of $[M : \mathbb{E}]$. Let ℓ be a prime divisor; then α is contained in the subextension of M of degree d/ℓ over $\mathbb{E}$ if and only if the Frobenius automorphism of this subextension leaves α invariant. ♦

Algorithm 4.10 (Finding a generator for a compositum of t finite fields in a given multiplicative subgroup.)
Input: finite fields $\mathbb{E} \subseteq \mathbb{F}$, with $|\mathbb{E}| = q$, and nonzero elements $\alpha_1, \ldots, \alpha_t$ in $\mathbb{F}$.
Output: integers $x_1, \ldots, x_t$ such that

$$\prod_{i=1}^t \alpha_i^{x_i} \text{ generates the composite field } \mathbb{E}(\alpha_1, \ldots, \alpha_t) \text{ over } \mathbb{E}.$$

```
1: [Precomputation] Compute $\beta^q$, where $\beta$ is the given generator for $\mathbb{F}$ over $\mathbb{E}$.
2: Put $d = 1$. Then, for $i = 1, \ldots, t$ do:
    a: [Degree of $\alpha_i$] Using Algorithm 4.5, compute the degree $e_i$ of $\alpha_i$ over $\mathbb{E}$.
    b: [Relative degree] Put $f_i = e_i / \gcd(d, e_i)$, and replace $d$ by $df_i$.
3: Factor $d$ [now the degree of the composite field over $\mathbb{E}$] into primes; let $P$ be the
   set of prime divisors of $d$.
4: [Iterate over basis elements] For $x_1 = 0, \ldots, f_1 - 1, \ldots, x_t = 0, \ldots, f_t - 1$ do:
    a: [Compute basis element] Put $\gamma = \prod_{i=1}^t \alpha_i^{x_i}$.
    b: [Apply Frobenius maps of maximal subfields] For all $\ell \in P$, put

           $\gamma_\ell = \prod_{i=1}^t \alpha_i^{x_i q^{d/\ell}}$.

       Use the Galois conjugates of the $\alpha_i$ computed in Step 2a.
    c: [Not in subfield?] If for all $\ell \in P$ we have $\gamma \neq \gamma_\ell$, output $(x_1, \ldots, x_t)$ and
       terminate the algorithm.
```

Proposition 4.11 *Algorithm 4.10 is correct and deterministic. Its running time is bounded by the cost of $\tilde{O}(e(\log q) + te^2)$ operations in $\mathbb{E}$, where $e = [\mathbb{F} : \mathbb{E}]$ and $q = |\mathbb{E}|$, and $O(e)$ other bit operations. The resulting integers x_i are nonnegative and less than e.*

Proof. The correctness of the given method follows from Lemmata 4.7, 4.8, and 4.9. The loop in Step 4 is abandoned as soon as one basis element is found to satisfy the requirements for being a field generator.

The bounds on the resulting integers x_i follow also from Lemma 4.8.

Write $e = [\mathbb{F} : \mathbb{E}]$; we have $d \leq e$. Step 1 spends $O(\log q)$ operations in $\mathbb{F}$, hence $\tilde{O}(e \log q)$ operations in $\mathbb{E}$, Step 2 spends $\tilde{O}(te^2)$ operations in $\mathbb{E}$ by Proposition 4.6. Step 3 spends $O(e)$ bit operations for factoring d. Steps 4a to 4c are executed at most e times, and in each iteration spend $O\left(t(\log e) + (\log e)^2\right)$ multiplications in $\mathbb{F}$. Namely, we have $|P| = O(\log e)$, and raising all α_i, as well as their chosen conjugates, to powers x_i hence takes $O((\log e) \sum_i O(\log f_i)) = O((\log e)^2)$ multiplications; an additional $O(t(\log e))$ are used for multiplying these powers together. It follows that the whole of Step 4 can be done using $\tilde{O}(te)$ multiplications in $\mathbb{F}$, hence $\tilde{O}(te^2)$ multiplications in $\mathbb{E}$. ♦

Remarks. Put in a certain order, each of the basis elements computed in Step 4a is derived from one of its precedessors by multiplying with just one of the α_i. *Mutatis mutandis*, the same holds for the Frobenius images of these elements in Step 4b. Therefore, at the cost of some administration, we could complete Step 4 as a whole using just $O(e \log e)$ multiplications.

It could also be useful to discard α_i if f_i turns out to be 1; in other words, to discard α_i if it is already contained in M_{i-1}. This ensures that the degree of M_i is at least twice that of M_{i-1}, for each i, and hence the number t of generators will be $O(\log e)$.

Proof of Theorem 4.1. Algorithm 4.10 claims to compute the desired generator of the composite field. By Proposition 4.11, it is correct and deterministic, and finishes in time polynomial in t and $\log |\mathbb{F}|$. ⧫

4.4 Finding nth power generators

This Section will prove Theorem 4.2, using the results from the previous Section. In fact, we will give a more general algorithm than needed for Theorem 4.2 (Algorithm 4.14 below). Namely, Algorithm 4.14 will work with a given subfield $\mathbb{E}$ as base field, so not only over the prime field.

Thus, let $\mathbb{E} \subseteq \mathbb{F}$ be an extension of finite fields, where $|\mathbb{E}| = q$, and let n be a positive integer. Then Algorithm 4.14 computes an element α such that $\mathbb{E}(\alpha^n)$ is equal to the field obtained by adjoining all nth powers in $\mathbb{F}$ to $\mathbb{E}$; this also provides a proof that such an element exists.

By Proposition 2.2, the set K of sums of nth powers in $\mathbb{F}$ is a subfield of $\mathbb{F}$; it is, of course, equal to the field obtained by adjoining all nth powers in $\mathbb{F}$ to its prime field. It is easy to see that we have $\mathbb{E}(\alpha^n) = \mathbb{E} \cdot K$, where α is as above. By Proposition 2.2(iii), the field K can be different from $\mathbb{F}$ only if we have $q \leq n^2$. *A fortiori*, the same holds for the field $\mathbb{E} \cdot K$ containing K.

Outline. In the situation where $q^e \leq n^2$, we are in a position to enumerate *all* nth powers in $\mathbb{F}^*$. We could then use Algorithm 4.10 to determine an nth power generator for the subfield of $\mathbb{F}$ generated over $\mathbb{E}$ by these nth powers, but as $\mathbb{F}^{*n}$ is a cyclic group, we see that the output of Algorithm 4.10 is already among the enumerated elements. Thus, a simpler method is to find an element of $\mathbb{F}^{*n}$ of maximal degree over $\mathbb{E}$.

In all other situations, the subfield generated over $\mathbb{E}$ by the nth powers is equal to $\mathbb{F}$. Now to proceed, we want the base field $\mathbb{E}$ to have at least $n+1$ elements, for reasons that will be explained presently.

Thus, first assume that $\mathbb{E}$ is "small", that is, $\mathbb{E}$ has at most n elements. A simple but effective method to enlarge $\mathbb{E}$ is to find n distinct nth powers in $\mathbb{F}^*$ and to adjoin these to $\mathbb{E}$; that will ensure that afterwards we have $|\mathbb{E}| > n$. To obtain enough distinct elements we enumerate at most $(n-1)^2+1$ elements of $\mathbb{F}^*$ and examine their nth powers.

We can "compose" the n distinct nth powers into just one by calling the compositum algorithm 4.10. This will give us an element $\alpha \in \mathbb{F}$ such that $\mathbb{E}(\alpha^n)$ has more than n elements. In case $\mathbb{E}$ already has enough elements, we simply take $\alpha = 1$.

We may now assume that $\mathbb{E}$ is "large"; and for such fields $\mathbb{E}$ we can easily find some nth powers that together generate $\mathbb{F}$ over $\mathbb{E}$, as the following Lemma and its Corollary show.

Lemma 4.12 *Let β generate $\mathbb{F}$ over $\mathbb{E}$, and suppose $c_0, \ldots, c_n$ are distinct elements of $\mathbb{E}$. Then if $\beta \notin \mathbb{E}$, we also have $(\beta + c_i)^n \notin \mathbb{E}$ for at least one i with $0 \le i \le n$.*

Proof. Assume the contrary; put $q = |\mathbb{E}|$. Then for all i, we have $(\beta + c_i)^{nq} = (\beta + c_i)^n$, so $(\beta + c_i)^{q-1}$ is an nth root of unity in $\mathbb{F}$, of which there are at most n. By the pigeonhole principle, there exist i and j with $0 \le i < j \le n$ such that $(\beta+c_i)^{q-1} = (\beta+c_j)^{q-1}$, which implies $\frac{\beta+c_i}{\beta+c_j} \in \mathbb{E}$. But this is a contradiction, because β is not in $\mathbb{E}$, and we have $c_i \neq c_j$. ♦

Corollary 4.13 *With the same assumptions as in the Lemma, the elements $(\beta+c_i)^n$ (for $i = 0, \ldots, n$) together generate $\mathbb{F}$ over $\mathbb{E}$.*

Proof. Retaining the same elements c_i, apply the Lemma successively to all maximal subfields of $\mathbb{F}$ containing $\mathbb{E}$ (in the role of $\mathbb{E}$). It follows that no such field contains all the elements $(\beta + c_i)^n$. Therefore these elements generate the whole field $\mathbb{F}$. ♦

We apply this construction to $\mathbb{F}/\mathbb{E}(\alpha^n)$, where α is defined as above. Once we have obtained the c_i, then with a second call to Algorithm 4.10, we "compose" the elements $\beta + c_i$, for $i = 0, \ldots, n$, together with α, to find a single element α whose nth power generates $\mathbb{F}$ over $\mathbb{E}$. This solves our problem.

Algorithm 4.14 (Finding an nth power generator for a finite field.)
Input: finite fields $\mathbb{E} \subseteq \mathbb{F}$, with $|\mathbb{E}| = q$ and $[\mathbb{F} : \mathbb{E}] = e$, and a positive integer n dividing $q^e - 1$.
Output: an element $\alpha \in \mathbb{F}$ such that $\mathbb{E}(\alpha^n)$ is equal to the field K generated over $\mathbb{E}$ by all nth powers in $\mathbb{F}$. [We have $K = \mathbb{F}$ whenever $q^e > n^2$.]

```
1: [F small?] If $q^e \le n^2$ then:
    a: [Find powers] Put $t = \frac{q^e-1}{n}$ and let $\gamma_1, \ldots, \gamma_t$ be elements of $\mathbb{F}^*$ whose $n$th powers are all distinct.
    b: [Find degree] Compute $\beta^q$, where $\beta$ is the given generator of $\mathbb{F}$ over $\mathbb{E}$. Using Algorithm 4.5, find $i$ with $1 \le i \le t$ such that $[\mathbb{E}(\gamma_i^n) : \mathbb{E}]$ is maximal.
    c: [Result] Output $\gamma_i$ and terminate.
```

```
2: [$\mathbb{E}$ small?] Put $\alpha = 1$. If $q \le n$ then:
    a: [Find powers] Enumerate elements of $\mathbb{F}$ until we have found $n$ nonzero
       elements $\gamma_1, \ldots, \gamma_n$ whose $n$th powers are all distinct.
    b: [Compose] Apply Algorithm 4.10 to $\mathbb{E}$, $\mathbb{F}$, and $\gamma_1^n, \ldots, \gamma_n^n$; let $x_1, \ldots, x_n$
       be the result. Replace $\alpha$ by $\gamma_1^{x_1} \cdots \gamma_n^{x_n}$. [Now $\mathbb{E}(\alpha^n)$ has more than $n$
       elements.]
3: [Not done yet?] If $[\mathbb{E}(\alpha^n) : \mathbb{E}] < [\mathbb{F} : \mathbb{E}]$ then:
    a: [Enumerate] Compute $n+1$ arbitrary distinct elements $c_0, \ldots, c_n$ of $\mathbb{E}(\alpha^n)$.
    b: [Compose] Apply Algorithm 4.10 to $\mathbb{E}$, $\mathbb{F}$, and $\alpha^n, (\beta+c_0)^n, \ldots, (\beta+c_n)^n$,
       where $\beta$ is the given generator for $\mathbb{F}$ over $\mathbb{E}$; let $z, y_0, \ldots, y_n$ be the result.
       Replace $\alpha$ by
           $\alpha^z (\beta + c_0)^{y_0} \cdots (\beta + c_n)^{y_n}$.
4: [Result] Output $\alpha$ and terminate.
```

Remarks. The degree in Step 3 is computed already by Algorithm 4.10, although formally it is not given as output.

If we compute the γ_i in Step 2, we can reuse them in Step 3a by putting $c_i = \gamma_i^n$ for $i \ge 1$, and $c_0 = 0$.

Proposition 4.15 *Algorithm 4.14 is correct and deterministic. It can be run using $\tilde{O}(n^2 e + ne^2 + e \log q)$ operations in $\mathbb{E}$, where $e = [\mathbb{F} : \mathbb{E}]$ and $q = |\mathbb{E}|$.*

Proof. The correctness of Algorithm 4.14 follows from the discussion above.

Step 1a and Step 2a each take at most $O(n^2 \log n)$ operations in $\mathbb{F}$, because every equation $x^n = a$ has at most n solutions, and we have $t < n$ in Step 1a. Each of the two calls to Algorithm 4.10, and also Step 1b, which is a subset of Algorithm 4.10, takes $\tilde{O}(e(\log q) + ne^2)$ operations in $\mathbb{E}$ by Proposition 4.11, while the computations of the new values for β in Steps 2b and 3b take $O(ne)$ operations in $\mathbb{F}$, due to the x_i, y_i, and z being bounded by $e = [\mathbb{F} : \mathbb{E}]$. The bound in the Proposition follows by the assumption that one operation in $\mathbb{F}$ takes $\tilde{O}(e)$ operations in $\mathbb{E}$. ♦

Proof of Theorem 4.2. Write $q = |\mathbb{F}|$. If n does not divide $q-1$, we replace n by $\gcd(n, q-1)$ as described in Section 2.2. Now Algorithm 4.14, applied to the extension $\mathbb{F}_p \subseteq \mathbb{F}$, generates the subfield K of sums of nth powers in $\mathbb{F}$ by means of adjunction of β^n to $\mathbb{F}_p$. By Proposition 4.15, this algorithm is correct and deterministic, and finishes in time polynomial in n and $\log q$. ♦

Generalisations. The multiplicative compositum algorithm 4.10 is valid for a finite cyclic extension L/K of arbitrary fields, provided:

(i) we can compute the degree of an element of L over K (efficiently);

(ii) we can determine (efficiently) whether a field element is contained in one or more intermediate fields (i.e., fields M with $K \subseteq M \subseteq L$). This condition is satisfied, for example, if we know a generator of the Galois group of L over K.

It follows that Algorithm 4.10 should work for cyclic extensions of number fields, for example, as well as for finite fields.

We note that Lemma 4.12 and its Corollary are valid for arbitrary finite cyclic Galois extensions. (The same proof works, except that one should replace the map $x \mapsto x^q$ by a generator of the Galois group.) It follows that nth power generators can be computed for such extensions by Algorithm 4.14.

As regards Lemma 4.7, it is an interesting question from representation theory under which conditions every basis contains a ring generator, if we consider noncyclic extensions of fields or more general rings. For example, if L/K is a Galois extension with Galois group V_4, then one can easily write down a basis for L over K such that no basis element generates L; this follows because the vector space sum of the three quadratic intermediate fields is equal to L. Also, if K and L are number fields, it is natural to restrict attention to *integral bases*. See [51] for some results in this area.

Chapter 5

Sums of like powers

5.1 Introduction and results

We consider a finite field $\mathbb{F}$. Given a positive integer n, can we write any given element of $\mathbb{F}$ as a sum of nth powers of elements of $\mathbb{F}$? And if so, how many such powers are needed?

This problem, known as Waring's problem for finite fields by analogy to its classical formulation with respect to the integers, has known active research in the 20th century. Some elementary results are recalled in Proposition 2.2 (especially (vi)), but in the meantime better bounds have been obtained in the cases where n is small with respect to $|\mathbb{F}|$; see Section 2.5 for more details.

What concerns us here is the question of *actually computing* representations of given elements as sums of nth powers. The main result is the following.

Theorem 5.1 *There exists a deterministic algorithm that, given a finite field $\mathbb{F}$ with q elements, a positive integer n and a nonzero element a of $\mathbb{F}$, determines elements $x_1, \ldots, x_n$ of $\mathbb{F}$ such that*

$$a = \sum_{i=1}^{n} x_i^n, \tag{5.2}$$

or correctly asserts that a is not a sum of nth powers in $\mathbb{F}$, in time polynomial in n and in $\log q$.

I will propose such an algorithm below (Algorithm 5.22). As far as I know, this is the first efficient deterministic algorithm to write finite field elements as sums of powers.

After the results of Chapter 4, representing a nonzero element a of $\mathbb{F}$ as a sum of nth powers is easy; we must, however, reduce the number of terms in the sum to at most n, as claimed in Theorem 5.1. Section 2 shows how to do this; the basic method obtained there (Algorithm 5.9) does not yet admit a polynomial running time bound.

Section 3 presents three auxiliary techniques to improve the asymptotic complexity of Algorithm 5.9. The first of these techniques is concerned with the right choice of data structures, while the other two provide good representations of elements of the prime field $\mathbb{F}_p$ as sums of nth powers.

We comment briefly on the two latter methods, which are of independent interest.

The second technique consists of a greedy algorithm that subtracts repeatedly from a positive integer a the greatest possible integer nth power; this is useful as long as $a \gg n^n$. This is obviously applicable to elements of a prime field $\mathbb{F}_p$, considered as integers between 0 and $p-1$, if p is very large compared to n.

The third technique expands a as $a = \sum_{i=0}^{d} a_i (t/s)^i$, where s and t are integers with $1 \leq s < t \leq n+1$, so effectively the element a is "expanded on the base t/s". The "digits" a_i satisfy $0 \leq a_i \leq n$. This extension of the usual representation of integers on an integer base seems to be new. The application to our problem is as follows: by Selective Root Extraction we can determine s and t such that an nth root of t/s in $\mathbb{F}_p$ can be computed efficiently and deterministically. This technique is useful whenever $a > n$.

Section 4 then gives an algorithm that proves Theorem 5.1; it is an improved version of Algorithm 5.9, incorporating the techniques from Section 3.

Finally, in Section 5, we consider representations of *zero* as a sum of nth powers. These may be obtained by applying Algorithm 5.22 to the element -1, which is obviously a sum of nth powers for all n in every finite field. However, making use of some properties of roots of unity, we can construct for this problem a faster version of Algorithm 5.22. The resulting running time bound, given in the following Theorem, bridges the gap in complexity between probabilistic and deterministic algorithms in this case.

Theorem 5.3 *There exists a deterministic algorithm which, given a finite field $\mathbb{F}$ with q elements and a positive integer n, determines elements $x_0, \ldots, x_n$ of $\mathbb{F}$, with $x_0 \neq 0$, such that*

$$\sum_{i=0}^{n} x_i^n = 0, \tag{5.4}$$

using $\tilde{O}(n^2 (\log q)^2)$ bit operations to finish.

Conventions. In this Section, we suppose that elements of $\mathbb{F}_p$ are represented by integers between 0 and $p-1$ inclusive, and we will move back and forth between integers and field elements without notice.

Because the algorithms given in this Chapter use both operations in the ring of integers and in various finite fields, we give the complexity bounds in terms of *bit operations.* We use the result that elements of a finite field of q elements can be added and multiplied using $\tilde{O}(\log q)$ bit operations (cf. Section 1.5).

A logarithmic lemma. All logarithms occurring in this Chapter are natural logarithms. The following lemma will be used several times.

Lemma 5.5 *For all $x \in \mathbb{R}$, $x > 1$, we have* $\dfrac{1}{x} < \log \dfrac{x}{x-1} < \dfrac{1}{x-1}$.

Proof. We have $\log \dfrac{x}{x-1} = -\log\left(1 - \dfrac{1}{x}\right) = \sum_{i \geq 1} \dfrac{1}{i \cdot x^i}$, where the Taylor series converges for all $x \in \mathbb{R}$ with $x > 1$. The Lemma now follows by the obvious inequalities

$$\frac{1}{x} \leq \sum_{i \geq 1} \frac{1}{i \cdot x^i} \leq \sum_{i \geq 1} \frac{1}{x^i} = \frac{1}{x-1}.$$

♦

5.2 The main algorithm; basic version

This section explains the main algorithm whose existence will prove Theorem 5.1. The basic version described here, however, does not yet admit a polynomial running time bound. Therefore we postpone the investigation into the complexity of the method to Section 5.4.

Let $\mathbb{F}$ be a finite field with q elements, characteristic p, and extension degree e over its prime field. We are given a positive integer n and a nonzero element $a \in \mathbb{F}$, and we want to write a as a sum of nth powers, using at most n terms. The case where $a = 0$ will be treated in Section 5.5.

Initialisation. The first step is to generate the subfield K of all sums of nth powers in $\mathbb{F}$ by an element that is itself an nth power. By Proposition 2.2, such a generator always exists; furthermore, we have $K = \mathbb{F}$ whenever $n^2 \leq q$. Now if we have such an element α^n, we can write every $a \in K^*$ as

$$a = \sum_{i=0}^{f-1} a_i (\alpha^i)^n, \tag{5.6}$$

where $f = [K : \mathbb{F}_p]$, and the a_i, elements of the prime field, are interpreted as integer coefficients between 0 and $p-1$, and are not all zero.

Now if we try to write a given element $a \in \mathbb{F}$ in the form (5.6), we will succeed if and only if a is in K — that is, if and only if a is a sum of nth powers in $\mathbb{F}$.

The multisection stage. The representation (5.6), properly viewed, writes a as a sum of M nth powers, where $M = \sum_{i=0}^{f-1} a_i$. We want to reduce this to at most n powers; this is done as follows.

Divide the sequence of M terms into $n+1$ subsequences, all having roughly the same number of components. Next, form the $n+1$ sums S_1 up to S_{n+1}, where S_j is the sum of the nth powers of all terms contained in the first j subsequences.

If any of the S_j is zero, we immediately discard all the corresponding terms from our representation; note that after this operation, we still have a representation of our a as a sum of nth powers, but with fewer terms. If not, we apply Selective Root Extraction (Algorithm 3.12) to the sequence $(S_1, \ldots, S_{n+1})$ and obtain

$$S_l = \beta^n S_k$$

for some $\beta \in \mathbb{F}$ and some integers k and l with $1 \leq k < l \leq n+1$. Therefore, if we multiply all terms in the first k subsequences by β and discard all terms in the $(k+1)$th up to lth sequences, the sum of the nth powers of all terms is unchanged.

In both cases, the number of terms M will drop by a factor of about $\frac{n+1}{n}$ at worst. The trick is applicable as long as we have $M \geq n+1$; hence we will end up having $M \leq n$, as desired.

The reduction step is embodied in the following algorithm. Note that if $M \leq n$, the algorithm is still valid, although the output might be the same as the input; and also that the empty sequence is returned in case $\sum_{i=1}^{M} y_i^n = 0$.

Algorithm 5.7 (Power sum reduction.)
Input: a finite field $\mathbb{F}$ with q elements, a positive integer n dividing $q-1$, and a sequence $(y_1, \ldots, y_M)$, with $M \geq 0$, of nonzero elements of $\mathbb{F}$.
Output: a sequence $(z_1, \ldots, z_N)$, with $N \geq 0$, of nonzero elements of $\mathbb{F}$ such that

(i) $\sum_{i=1}^{M} y_i^n = \sum_{i=1}^{N} z_i^n$;
(ii) $N \leq \frac{n}{n+1}(M+1)$.

```
1: [Form subsequences] Find integers $Q$ and $R$, with $0 \leq R < n+1$, such that
   $M = Q\cdot(n+1)+R$. For $j = 1, \ldots, R$, put $m_j = j(Q+1)$; for $j = R+1, \ldots, n+1$,
   put $m_j = R(Q+1) + (j-R)Q$.
   [The $j$th subsequence, for $j \geq 2$, contains the terms $y_i$ for $i = m_{j-1}+1, \ldots, m_j$;
   and we have $m_{n+1} = M$.]
2: [Partial sums] Compute $S_1, \ldots, S_{n+1}$ by $S_j = \sum_{i=1}^{m_j} y_i^n$ $(1 \leq j \leq n+1)$.
3: [Zero sum?] If $S_k = 0$ for some $k$, take the largest such $k$, output $(y_i)_{i=m_k+1}^{M}$
   and terminate.
4: [Force equal sums] Using Algorithm 3.12 with arguments $(S_j)_{j=1}^{n+1}$, find integers
   $k$ and $l$ with $1 \leq k < l \leq n+1$ and an element $\beta \in \mathbb{F}^*$ such that $S_l = \beta^n S_k$.
5: [Result] Output the concatenation of $(\beta y_i)_{i=1}^{m_k}$ and $(y_i)_{i=m_l+1}^{M}$.
```

Lemma 5.8 *Algorithm 5.7 is correct and deterministic. It finishes using* $\tilde{O}(M \log n + n \log q + (\log q)^2)$ *operations in* $\mathbb{F}$.

Proof. We divide the sequence (y_i) of M terms into subsequences whose lenghts are as follows:

$$\underbrace{Q+1, Q+1, \ldots, Q+1}_{R \text{ times}},\ \underbrace{Q, \ldots, Q}_{n+1-R \text{ times}}\ .$$

As we have no control over which subsequence will be discarded, this ensures that we profit equally from all arising situations.

Now either in Step 3 or in Step 5, we discard at least Q terms. As Q is not less than $\frac{M-n}{n+1}$, it follows that

$$N \le M - Q \le M - \tfrac{M-n}{n+1} = \tfrac{n}{n+1}\,(M+1)\,.$$

The running time of Algorithm 5.7 is bounded by $O(M \log n)$ operations in $\mathbb{F}$ for computing the partial sums S_j, and a further $\tilde{O}(n \log q + (\log q)^2)$ operations in Step 4, by Proposition 3.13. ♦

Main algorithm. The following algorithm puts together the pieces we discussed above.

Algorithm 5.9 (Power sum representation; basic version.)
Input: a finite field $\mathbb{F}$ with q elements and characteristic p, a positive integer n dividing $q-1$, and a nonzero element a of $\mathbb{F}$.
Output: "no solution", or a sequence $(x_i)_{i=1}^n$ in $\mathbb{F}$ such that $a = \sum_{i=1}^n x_i^n$.

```
1: [Find field generator] Apply Algorithm 4.14 with arguments $\mathbb{F}_p$, $\mathbb{F}$, and $n$ to
   find $\alpha$ such that $\mathbb{F}_p(\alpha^n)$ is equal to the set of all sums of $n$th powers in $\mathbb{F}$. Let
   $f$ denote the degree of $\alpha^n$ over $\mathbb{F}_p$.
2: [Represent $a$ on basis] Let $(a_0, \ldots, a_{f-1})$ be the coefficient vector of $a$ when
   represented on the basis $1, \alpha^n, \alpha^{2n}, \ldots, \alpha^{(f-1)n}$. If $a$ is not a linear combination
   of these elements, output "no solution" and terminate.
3: [Form initial sequence] Let $(y_i)_{i=1}^M$ be the sequence consisting of $a_i$ components
   $\alpha^i$, for $i = 0, \ldots, f-1$.
4: [Ready?] While $M > n$ do:
    a: [Reduce sequence] Replace $(y_i)_{i=1}^M$ by the output of Algorithm 5.7 applied
       to it.
5: [Result] Output $(y_i)_{i=1}^M$, followed by $(0)_{i=M+1}^n$.
```

Proposition 5.10 *Algorithm 5.9 is deterministic. It returns a correct solution whenever one exists, and "no solution" otherwise.*

Proof. If we apply Algorithm 4.14 with base field $\mathbb{F}_p$, the resulting field $\mathbb{F}_p(\alpha^n)$ is equal to the set of all sums of nth powers in $\mathbb{F}$. Therefore, the given element a is such a sum if and only if it is an element of $\mathbb{F}_p(\alpha^n)$. This can be checked by trying to represent a on the basis $1, \alpha^n, \alpha^{2n}, \ldots$ over $\mathbb{F}_p$, as is done in Step 2. If Step 2 fails, then a is not a sum of nth powers in $\mathbb{F}$ and the algorithm stops saying "no solution". If it is, we obtain a representation of the form (5.6), and the algorithm can proceed.

We prove termination of the loop in Step 4. By Lemma 5.8, the number of terms N of the sequence returned by Algorithm 5.7 satisfies

$$N \le \tfrac{n}{n+1}\,(M+1)\,. \tag{5.11}$$

Now if we have $M \ge n+1$, it follows that $N < M$, and thus eventually we will have $M \le n$. ♦

5.3 Improving the complexity

If we consider Algorithm 5.9 as it stands, it is easy to see that the running time cannot be polynomial. Namely, at the beginning of Step 4, the length M of the sequence of powers that we consider can be as large as $e(p-1)$, because the coefficients a_i computed in Step 2 are integers between 0 and $p-1$. Therefore Lemma 5.8 proves nothing better than that every iteration in Step 4 takes at most exponential time in $\log p$.

In this section, we give three techniques for improving the complexity of Algorithm 5.9.

Firstly, we can make use of the fact that the number of *distinct* terms in the sequence $(y_i)_{i=1}^M$ is very small: initially, it is at most e. Now summing up a large number of equal terms is better done by means of multiplication by an integer than by repeated addition.

Secondly and thirdly, we can try to reduce the size of the integers a_i, thus reducing the number of terms M at the cost of a slightly higher number of *distinct* terms.

The two auxiliary algorithms (Algorithms 5.14 and 5.19) that we present below are concerned with the representation of elements of *prime fields* as sums of nth powers. Obviously, for any $a \in \mathbb{F}_p$ and any positive integer n, we have

$$a = \sum_{i=1}^{a} 1^n. \tag{5.12}$$

This sum has a terms; we want to decrease this number. Algorithm 5.14 is useful when $a \gg n^n$; Algorithm 5.19 works whenever $a > n$.

After applying these two algorithms, the number of terms M will even be polynomial in $\log\log q$; however, there will appear a polynomial dependence of M on the degree n.

5.3.1 Equal and distinct terms

We begin by showing how Algorithm 5.7 can make use of the fact that many terms in the input sequence are equal.

Lemma 5.13 *Consider the sequence $(y_i)_{i=1}^M$ in the input of Algorithm 5.7. If the y_i are arranged in $\tilde{M}$ blocks of equal terms, Algorithm 5.7 can be run using $\tilde{O}((\log n)\tilde{M} + n\log q + (\log q)^2)$ operations in $\mathbb{F}$, and the resulting sequence $(z_i)_{i=1}^N$ will have the same form, with at most $\tilde{M}+1$ blocks.*

Proof. By forming the $n+1$ subsequences in Step 1, we may divide some of the blocks of equal terms into two or more; but at the cost of increasing $\tilde{M}$ by n, we may assume that every block is contained in one subsequence.

It follows that the partial sums S_j may be calculated using $O((\tilde{M}+n)\log n)$ operations in $\mathbb{F}$. A further $\tilde{O}(n\log q + (\log q)^2)$ operations are used in Step 4, by Proposition 3.13. This gives the bound on the running time.

To form the new sequence $(z_i)_{i=1}^N$, some blocks are multiplied by β, some are discarded and some are kept. In terms of the blocks originally supplied, we see that some blocks lose some or all of their members, some are kept, some are multiplied by β, and at most one initial block has only *some* of its members multiplied by β, while the others remain the same. It follows that we can arrange the sequence (z_i) in the same form as the input sequence, at no additional cost, except that we may need one more block. ⧫

5.3.2 Sums of powers in the integers

Next we consider trying to write a given $a \in \mathbb{F}_p$ as a sum of nth powers in $\mathbb{Z}$, where only positive powers are allowed. Now it is known that for any n, there exists a constant $g(n)$ such that every integer can be written as a sum of at most $g(n)$ such powers; this constant $g(n)$, however, grows exponentially with n (see [58] and [28, Chapter XXI] for a discussion). As an example, to represent $2^n - 1$ as a sum of nth powers we obviously need $2^n - 1$ terms.

Therefore, the following greedy algorithm repeatedly subtracts from a the largest nth power that is smaller than a, until a drops below n^n, from where the number of terms would become too large.

Algorithm 5.14 (Sum of integer powers, approximately.)
Input: positive integers a and n.
Output: a nonnegative integer $\tilde{a}$ and a nonincreasing, possibly empty sequence $(r_i)_{i=1}^m$ of positive integers such that

$$\tilde{a} \le n^n, \qquad a = \textstyle\sum_{i=1}^m r_i^n + \tilde{a}.$$

```
1: [Initialise] Put $m = 0$.
2: [Small enough?] While $a > n^n$ do:
    a: [Compute root] Replace $m$ by $m + 1$ and compute $r_m$ as $\lfloor \sqrt[n]{a} \rfloor$.
    b: [Decrease $a$] Replace $a$ by $a - r_m^n$.
3: [Result] Output $a$ and $(r_i)_{i=1}^m$.
```

Remark. The sequence (r_i) can be made *strictly* decreasing if we halt the algorithm as soon as $a < 2(n/\log 2)^n$. This is easily proved by considering for which integers a we have $r^n > a/2$ (where $r = \lfloor \sqrt[n]{a} \rfloor$).

Lemma 5.15 *Algorithm 5.14 is correct and deterministic. The number m of terms satisfies $m \le n \log\log a + 2$. The algorithm uses $\tilde{O}(n \log a)$ bit operations.*

Proof. It is clear that the algorithm will produce a sequence with the required properties. We now prove the desired inequality on m.

Let a be an integer, and let $r = \lfloor \sqrt[n]{a} \rfloor$. We have

$$(x+1)^n - x^n = \int_x^{x+1} nt^{n-1}\,dt < n(x+1)^{n-1}$$

for any real $x \geq 0$; we apply this to $x = \sqrt[n]{a} - 1$, which is less than r, and find

$$a - r^n < (x+1)^n - x^n \leq n(x+1)^{n-1} = na^{\frac{n-1}{n}}.$$

Now writing $a_0 = a$, $r_i = \lfloor \sqrt[n]{a_{i-1}} \rfloor$, and $a_i = a_{i-1} - r_i^n$ for $i \geq 1$, we find

$$a_i \leq na_{i-1}^{\frac{n-1}{n}} \leq \ldots \leq n^{1+\frac{n-1}{n}+\ldots+\left(\frac{n-1}{n}\right)^{i-1}} a_0^{\left(\frac{n-1}{n}\right)^i},$$

hence

$$a_i \leq n^n a_0^{\left(\frac{n-1}{n}\right)^i} \tag{5.16}$$

by summing the geometric series.

Suppose now that $i \geq n \log\log a_0$. Then $\log\log a_0 + i \log \frac{n-1}{n} < 0$ by Lemma 5.5, and we obtain

$$a_0^{\left(\frac{n-1}{n}\right)^i} < \mathrm{e},$$

so $a_i < \mathrm{e} \cdot n^n$ after i iterations (here $\mathrm{e} = \exp(1)$). Now continuing for two more iterations, we will have $a_{i+2} < (\mathrm{e}-2)n^n < n^n$, and hence the algorithm will terminate.

We must comment on the complexity of Step 2a. The running time of an algorithm for computing integer roots is estimated in detail by Bernstein in [8, Section 8], but only time spent in multiplication of integers in included in the estimates. However, the cost of Bernstein's algorithm, which consists of some bisection steps followed by a Newton iteration, is obviously dominated by the multiplication time. From a sequence of simple verifications that is too long to include here, we derive from Bernstein's estimates a cost of $\tilde{O}((\log n)^3 + \log a)$ bit operations for Step 2a.

For the whole Algorithm 5.14, we obtain a bound of $\tilde{O}(n(\log\log a)((\log n)^3 + \log a))$ bit operations, which is $\tilde{O}(n \log a)$. ♦

5.3.3 Expanding on a rational base

The final auxiliary algorithm can write any $a \in \mathbb{F}_p^*$ as a sum of nth powers with at most $n^2 \log \|a\| + 1$ terms, where $\|a\|$ denotes the unique integer representative of a in the set $\{1, 2, \ldots, p-1\}$. We use the following Lemma, which extends the usual representation of integers to a given base (say, 10, or 2) to an arbitrary *rational* base, while avoiding the use of infinite expansions. The reason for doing this is that among the elements $\{1, 2, \ldots, n+1\}$, considered as elements of $\mathbb{F}_p$ for some prime $p > n+1$, there exist two whose quotient we can express as an nth power in $\mathbb{F}_p$ by means of Selective Root Extraction (Algorithm 3.12). Therefore, in (5.18) below, we may assume that an nth root of t/s is known.

Lemma 5.17 *Given integers s and t with $0 < s < t$ and $(s,t) = 1$, every positive integer a can be represented as*

$$a = \sum_{i=0}^{d} c_i \left(\frac{t}{s}\right)^i, \tag{5.18}$$

where the "digits" c_i are integers and satisfy $0 \leq c_i \leq t-1$, and where we have

$$d \in \mathbb{Z}, \quad 0 \leq d \leq \frac{\log a}{\log(t/s)}, \quad c_d \neq 0.$$

Proof. Write $a = qt + r$ for integers q and r with $0 \leq r < t$. Then we have

$$a = qt + r = qs\frac{t}{s} + r,$$

and we can do the same procedure with qs instead of a. Proceeding until $a = 0$, we express a in the desired form.

Furthermore, from (5.18) it follows that $a \geq (t/s)^d$, which proves the bound on d. ♦

Remark. One can prove that representations of the form (5.18) are unique, even if the sum is not an integer. The present "(t/s)-adic" number system is new and should not be confused with expansions that simply subtract the greatest possible power of t/s from a; this usually results in an infinite expansion (see [40] for an overview of results). I found this system around the same time as the authors of [3]; the latter paper discusses in depth the properties of this representation as a dynamical system and as a formal language.

Algorithm 5.19 (Writing an integer on the base t/s.)
Input: positive integers a, t, and s such that $t > s$.
Output: a sequence $(c_i)_{i=0}^{d}$ such that (5.18) holds.

```
1: [Initialise] Let $i = 0$.
2: [Ready?] While $a > 0$ do:
    a: [Compute next digit] Write $a = q\,t + r$, with $0 \leq r \leq t-1$.
    b: [Decrease $a$] Put $c_i = r$ and $i = i + 1$, and replace $a$ by $qs$.
3: [Result] Put $d = i - 1$; output $c_0, c_1, \ldots, c_d$.
```

Lemma 5.20 *Algorithm 5.19 is correct and deterministic. It takes $\tilde{O}(s(\log a)^2)$ bit operations to finish. The resulting sequence has length $d + 1$, where d is bounded by $(s + 1)\log a$; the coefficients c_i satisfy $0 \leq c_i \leq t - 1$.*

Proof. The bound for d follows from combining Lemma 5.17 with Lemma 5.5, while we note that $t/s \geq \frac{s+1}{s}$; this also bounds the number of iterations of the algorithm. ♦

Remark. Comparably to the situation with radix conversion and the Euclidean algorithm, a more careful analysis will probably give a softly linear complexity bound for this algorithm.

5.4 The main algorithm; final version

We now give the final version of the algorithm for writing a nonzero element as a sum of nth powers, incorporating the techniques from the previous Section into Algorithm 5.9.

Let $\mathbb{F}$ be a given finite field with prime field $\mathbb{F}_p$. We now apply (only once) the Selective Root Algorithm 3.12 to the set $\{1, \dots, n+1\}$, and obtain integers s and t, with $1 \le s < t \le n+1$, such that t/s is written explicitly as an nth power in $\mathbb{F}_p$.

Then if we have some $a \in \mathbb{F}_p$, we apply first Algorithm 5.14 and then Algorithm 5.19 to find

$$a = \sum_{i=1}^{m} r_i^n + \sum_{i=0}^{d} c_i \left(\frac{t}{s}\right)^i, \tag{5.21}$$

for integers m and d, and with $r_i \in \mathbb{F}_p$ and $c_i \in \{0, \dots, n\}$.

Finally, to obtain the timings of Lemma 5.13, we need only choose the right data structures, upon which we will not comment any further. The following algorithm differs from Algorithm 5.9 only in that Step 3 was replaced by Steps 3 and 4 below.

Algorithm 5.22 (Representing $a \in \mathbb{F}^*$ as a sum of nth powers in $\mathbb{F}$.)
Input: a finite field $\mathbb{F}$ with q elements and characteristic p, a positive integer n dividing $q-1$, and a nonzero element a of $\mathbb{F}$.
Output: "no solution", or a sequence $(x_i)_{i=1}^n$ in $\mathbb{F}$ such that $a = \sum_{i=1}^n x_i^n$.

1: [Find field generator] Apply Algorithm 4.14 with arguments $\mathbb{F}_p$, $\mathbb{F}$, and n to find α such that $\mathbb{F}_p(\alpha^n)$ is equal to the set of all sums of nth powers in $\mathbb{F}$. Let f denote the degree of α^n over $\mathbb{F}_p$.

2: [Represent a on basis] Let $(a_0, \dots, a_{f-1})$ be the coefficient vector of a when represented on the basis $1, \alpha^n, \alpha^{2n}, \dots, \alpha^{(f-1)n}$. If a is not a linear combination of these elements, output "no solution" and terminate.

3: [Precomputations] Using Algorithm 3.12 with arguments $(1, 2, \dots, n+1)$, find integers s and t with $1 \le s < t \le n+1$ and an element $b \in \mathbb{F}_p$ such that $t/s = b^n$.

4: [Decrease coefficients] Apply Algorithm 5.14 and Algorithm 5.19 to all the a_i in turn, using the integers t and s computed in Step 3, writing a_i in the form (5.21).
Multiply each instance of the form (5.21) with the appropriate power of α, write out all the "digits" c_i as sums of 1's, and concatenate all the results, to find a nonnegative integer M and a sequence $(y_i)_{i=1}^M$ in $\mathbb{F}^*$ such that $a = \sum_{i=1}^M y_i^n$.

```
5: [Ready?] While $M > n$ do:
    a: [Reduce sequence] Replace $(y_i)_{i=1}^{M}$ by the output of Algorithm 5.7 applied to it.
6: [Result] Output $(y_i)_{i=1}^{M}$, followed by $(0)_{i=M+1}^{n}$.
```

Proposition 5.23 *Algorithm 5.22 returns a correct solution whenever one exists, and "no solution" otherwise. It is deterministic and its running time is* $\tilde{O}(n^2(\log q)^2 + n(\log q)^3)$ *bit operations.*

Remark. As Algorithm 5.22 uses operations in both $\mathbb{F}$ and $\mathbb{F}_p$, as well as integer arithmetic, we use bit operations to measure the total running time.

Proof. The correctness of Steps 3 and 4 follows from Lemmata 5.15 and 5.20. The remaining parts are correct by Proposition 5.10.

We bound the running time of Algorithm 5.22 as follows, bounding everywhere f by $e = [\mathbb{F} : \mathbb{F}_p]$. Step 1 takes $\tilde{O}(n^2e + ne + e\log p)$ operations in $\mathbb{F}_p$ by Proposition 4.15. Step 2 solves a linear system of e equations and f unknowns, which takes $O(e^3)$ operations in $\mathbb{F}_p$. Step 3 takes $\tilde{O}(n(\log p) + (\log p)^2)$ operations in $\mathbb{F}_p$ by Proposition 3.13.

By Lemma 5.20, Step 4 takes $\tilde{O}(en(\log p)^2)$ bit operations for f invocations of Algorithm 5.19 on integers below p. To this, by Lemma 5.15, we must add a cost of $\tilde{O}(en\log p)$ bit operations for f calls to Algorithm 5.14; this does not change the bound of $\tilde{O}(en(\log p)^2)$ operations.

For the resulting length M of the sequence (y_i) before Step 5, we use two different bounds, according as p is greater or less than n^n. If $p \le n^n$, we find $O(fn^2\log p) = O(n^2\log q)$ terms from Lemma 5.20. If $p > n^n$, we first have $O(fn\log\log p)$ terms, by Lemma 5.15; after that, Algoritm 5.19 is, in fact, applied to integers below n^n, so that Lemma 5.20 gives an additional $O(fn^2\log n^n) = O(en^3\log n)$ terms. Together, this makes $O(en(\log\log p + n^2\log n))$. As we have the better bound $e = \frac{\log q}{\log p} < \frac{\log q}{n\log n}$ in this case, we again find a "soft" bound of $\tilde{O}(n^2\log q)$ in the case where $p > n^n$.

We bound the number of iterations in Step 5 using (5.11) in the equivalent formulation

$$N - n \le \tfrac{n}{n+1}(M - n).$$

If we denote by $M_0, M_1, \ldots$ the subsequent values taken on by the sequence length M, we find by induction, for $i \ge 0$,

$$M_i \le (\tfrac{n}{n+1})^i(M_0 - n) + n < (\tfrac{n}{n+1})^i M_0 + n.$$

By Lemma 5.5, the right hand side decays to less than $n+1$ in $O(n\log M_0)$ iterations.

Substituting the initial bound for M, we find

$$O(n(\log n + \log\log q))$$

iterations, which is $\tilde{O}(n\log\log q)$.

Although no term of the sequence (y_i) is known to occur more than n times after Step 4, we can still benefit from the techniques given in Lemma 5.13, because we know that the number of *distinct* terms after Step 4 is $O(en \log p) = O(n \log q)$. (It is the number of "digits" given by Algorithm 5.19 plus the number of terms produced by Algorithm 5.14, if applicable.) By Lemma 5.13, this number may increase by at most the number of iterations; asymptotically speaking, this is no growth at all.

Therefore, each call to Algorithm 5.7 in Step 5 uses $\tilde{O}(n \log n \log q + n \log q + (\log q)^2)$ operations in $\mathbb{F}$, by Lemma 5.13, and Step 5 in total takes $\tilde{O}(n^2(\log q) + n(\log q)^2)$ operations in $\mathbb{F}$.

We add everything up, where we use the fact that $e \log p = \log q$, and the assumption that operations in $\mathbb{F}_p$ take $\tilde{O}(\log p)$, and operations in $\mathbb{F}$ take $\tilde{O}(\log q)$ bit operations. Thus, we obtain a bound of

$$\tilde{O}(n^2(\log q)^2 + n(\log q)^3)$$

bit operations, as claimed. ♦

Remarks. In Algorithm 5.7, which is called in Step 5a, many partial sums of $\sum y_i^n$ are formed, which are each tested for being zero. One might wonder if this event will ever occur. Now every subsum of the initial representation (5.6) is the unique representation of some nonzero element of $\mathbb{F}$ on some basis over $\mathbb{F}_p$ (namely, the power basis generated by α^n), and therefore cannot be zero. However, the use of Algorithm 5.19 for reducing the number of repeated terms in the sequence introduces the possibility of zero partial sums. An example is given by $(\frac{3}{2})^2 + \frac{3}{2} + 2 = \frac{23}{4}$, which is zero in $\mathbb{F}_{23}$. This is a subsum of $2 \cdot (\frac{3}{2})^2 + \frac{3}{2} + 2 = 8$, an example of a representation computed by Algorithm 5.19 applied with $t = 3$ and $s = 2$.

It is not necessary to apply all three techniques discussed above to obtain a polynomial time algorithm. In fact, the use of either Lemma 5.13 or Algorithm 5.19 by itself will suffice. On the other hand, using only Algorithm 5.14 will result in a running time that is polynomial in $\log q$, but at worst exponential in n.

As appears from the above proof, the use of Algorithm 5.14 in Step 4 does not improve the "soft" asymptotic complexity of the algorithm as a whole. The reason for this is that the calls to the Selective Root Algorithm 3.12 become the bottle neck in Algorithm 5.7. However, if we could assume that the degree e is bounded, for example, then we find a bound of $\tilde{O}(n \log \log q + n^3)$ for the initial number of terms M, which is obviously better than the bound of $\tilde{O}(n^2 \log q)$ used in the proof, whereas the cost of Algorithm 5.14 is negligible.

Proof of Theorem 5.1. Algorithm 5.22 claims to solve the problem posed in the Theorem, with the remark that if n does not divide $q-1$, we replace n by $\gcd(n, q-1)$ as described in Section 2.2. By Proposition 5.23, its output satisfies the bounds required by the Theorem and its running time is polynomial in n and $\log q$. ♦

5.5 The use of roots of unity

Next, we prove Theorem 5.3, hence we consider the case where we want a sequence of nth powers in a given finite field $\mathbb{F}$ whose sum is 0. Here we must allow for the use of $n+1$ terms, instead of n, in order to have solvability in all cases. On the other hand, this equation is already solvable over prime fields, and consequently there are no fields $\mathbb{F}$ over which a solution does not exist (cf. Theorem 2.3(i)).

A simple solution is calling Algorithm 5.22 with argument -1; viz., if $\sum_{i=1}^{n} x_i^n = -1$, then $\sum_{i=1}^{n} x_i^n + 1^n = 0$. However, we will obtain a better complexity for this situation than that of Algorithm 5.22. For this, we make use of the properties of roots of unity in $\mathbb{F}$.

Roots of unity. It is well known that, if ζ_ℓ is a primitive ℓth root of unity for some integer $\ell \geq 2$, we have

$$\sum_{i=0}^{\ell-1} \zeta^i = 0. \tag{5.24}$$

Thus, if we contrive to take an nth root of ζ_ℓ, for some ℓ not exceeding $n+1$, we will find a sum of nth powers that is zero and has at most $n+1$ terms, and we are done.

A simple but important example of this technique is the situation where n is odd: here we take $\ell = 2$, so we have $\zeta_\ell = -1$, and of course an nth root of -1 is -1 itself. This leads to an obvious solution of (5.4), namely

$$1^n + (-1)^n = 0.$$

If n is even, then, to apply the same technique for $\ell = 2$, we must obtain a 2^eth root of -1 for some positive integer e, and it is not known how to do this efficiently and deterministically.

For general n, the following Lemma shows when we can make use of (5.24).

Lemma 5.25 *Let ℓ be a prime, let n be a positive integer, and let $a \in \mathbb{F}^*$ be such that*

$$v_\ell(\operatorname{ord} a) > v_\ell(n).$$

Then there exists an integer x such that $(a^x)^n$ is a primitive ℓth root of unity.

Proof. Write $n = m \cdot \ell^f$, where $\ell \nmid m$ and $f \geq 0$. The order of a is divisible by ℓ^{f+1}, so some power a^x of a has order exactly ℓ^{f+1}. It follows that $(a^x)^n$ has order ℓ, as desired. ♦

The complexity of Selective Root Extraction. As Proposition 3.13 says, the complexity of the Selective Root Algorithm 3.12 in terms of $\log q$ is essentially cubic, when applied to general arguments. However, in the current situation, we need to apply this algorithm only to arguments a_i such that, for all primes ℓ dividing n, we have

$$v_\ell(\operatorname{ord} a_i) \leq v_\ell(n) \qquad (i = 0, \ldots, n). \tag{5.26}$$

Namely, if for some index i and prime ℓ this inequality fails, then we can take the 'side exit': Lemma 5.25 shows that the group generated by a_i contains some b such that b^n is an ℓth root of unity, and we find

$$1^n + b^n + b^{2n} + \ldots + b^{(\ell-1)n} = 0,$$

which solves the main equation (5.4). But if we can assume the bound (5.26), then the running time of Algorithm 3.12 drops to $O(n \log q)$ operations in $\mathbb{F}$, as shown by Lemma 3.14.

Main algorithm. To obtain the algorithm proving Theorem 5.3, we simply apply Algorithm 5.22 to the element -1, and then append a 1 to the resulting sequence. But first, we modify Algorithm 5.7 by inserting the following step between Steps 3 and 4, incorporating the 'side exit' discussed above. The input and output specifications of Algorithm 5.7 are not changed; the use of the side exit is limited to the case of writing -1 as a sum of nth powers.

Algorithm 5.27 (Extra step added to Algorithm 5.7.)

3*bis*: [Side exit] If $S_{n+1} = -1$, then for all primes ℓ dividing n, and for $i = 1, \ldots, n{+}1$:

- **a:** [Compute order] Write $q - 1 = u \cdot \ell^r$, where $\ell \nmid u$, and $n = m \cdot \ell^f$, where $\ell \nmid m$. Put $T = S_i^u$, and let ℓ^w be the order of T.
- **b:** [Order large enough?] If $w > f$, then let $b = T^{\ell^{w-f-1}}$, output $b, b^2, \ldots, b^{\ell-1}$ and terminate.

Lemma 5.28 *Algorithm 5.27 is correct and deterministic. The amount of operations in $\mathbb{F}$ needed for its execution is*

$$\begin{cases} \tilde{O}(M \log n + n \log q) & \textit{if its input satisfies } \sum_{i=1}^{M} y_i^n = -1; \\ \tilde{O}(M \log n + n \log q + (\log q)^2) & \textit{otherwise.} \end{cases}$$

Proof. Algorithm 5.27 is equal to Algorithm 5.7, unless its input satisfies

$$\sum_{i=1}^{M} y_i^n = -1;$$

this is equivalent to having $S_{n+1} = -1$. We will consider this situation.

The output of the modified algorithm 5.7 differs from the original version if, and only if, we find a prime ℓ dividing n and an index i between 1 and $n + 1$ such that the partial sum S_i has more factors in its order than n has. If these conditions are satisfied, then by Lemma 5.25 we know that b^n is an ℓth root of unity, where b is some integer power of S_i. To show that Step (3*bis*)b indeed computes the right element b, we compute

$$b^n = \left(T^{\ell^{w-f-1}}\right)^n = \left(T^{\ell^{w-1}}\right)^m.$$

As T has order ℓ^w, and as m is prime to ℓ, we find that $\operatorname{ord}(b^n)$ is exactly ℓ, as desired. The output of the modified algorithm is now the sequence of powers of b, which satisfies

$$b^n + b^{2n} + \ldots + b^{(\ell-1)n} = -1.$$

It follows that the nth powers of both the input and the output sequences sum to -1, and the correctness is proved.

As to the running time of the modified algorithm 5.7, we first note that the computations done in Step (3*bis*)a are also done in Algorithm 3.12, so that we do not need to count them. In Step (3*bis*)b, we perform $O(n + \log q)$ operations in $\mathbb{F}$.

Also, as said above, the call to Algorithm 3.12 in Step 4 takes now only $O(n \log q)$ operations in $\mathbb{F}$, by Lemma 3.14 and the bound (5.26). This completes the proof. ♦

One will notice that the precomputations in Step 3 of Algorithm 5.22 also contain a call to the Selective Root Algorithm 3.12. In order to obtain the desired complexity bound, we must take the "side exit" using roots of unity also in this step; this is the task of Step 1 of the main algorithm, which is now given.

Algorithm 5.29 (Representing $0 \in \mathbb{F}$ as a nontrivial sum of nth powers.)
Input: a finite field $\mathbb{F}$ with q elements and characteristic p, a positive integer n dividing $q-1$.
Output: a sequence $(x_i)_{i=0}^n$ in $\mathbb{F}$, with $x_0 \neq 0$, such that $\sum_{i=0}^n x_i^n = 0$.

```
1: [Side exit for Algorithm 5.22, Step 3]
   For all primes $\ell$ dividing $n$, and for $i = 1, \ldots, n+1$, do:
      a: [Compute order] Write $q - 1 = u \cdot \ell^r$, where $\ell \nmid u$, and $n = m \cdot \ell^f$, where
         $\ell \nmid m$. Put $I = i^u$ (considering the index $i$ as an element of $\mathbb{F}_p$), and let $\ell^w$
         be the order of $I$.
      b: [Order large enough?] If $w > f$, then let $b = I^{\ell^{w-f-1}}$, output $1, b, b^2, \ldots, b^{\ell-1}$
         followed by $(0)_{j=\ell+1}^n$, and terminate.
2: Using Algorithm 5.22 with the *modified version* 5.27 of Algorithm 5.7, compute
   $x_1, \ldots, x_n \in \mathbb{F}$ such that $\sum_{i=1}^n x_i^n = -1$.
3: Output (1) followed by $(x_i)_{i=1}^n$.
```

Proposition 5.30 *Algorithm 5.29 is correct and deterministic. Its running time is* $\tilde{O}(n^2(\log q)^2)$ *bit operations.*

Remark. As with Algorithm 5.22, the running time bound is given in terms of bit operations so that the cost of operations in various finite fields and also of integer arithmetic can be compared.

Proof. The correctness of Step 1 is proved in the same way as the correctness of Algorithm 5.27. By Theorem 2.3(i), a solution always exists, so the call to Algorithm 5.22 in Step 2 is always successful. This shows the correctness of Algorithm 5.29.

We provide a new estimate of the running time of Algorithm 5.22, this time using the modified version 5.27 of Algorithm 5.7. In the present case, the nth powers of the input to Algorithm 5.27 indeed sum to -1, so that the better estimate of $\tilde{O}(n \log q)$ operations of Lemma 5.28 applies. Also, Step 3 of Algorithm 5.22 takes $\tilde{O}(n \log q)$ operations, since all situations where it could take longer are detected in Step 1 of Algorithm 5.29.

All other operations in Algorithm 5.22, as well as Step 1 of Algorithm 5.29, take a linear number of finite field operations (linear in the logarithm of the field size). Hence the total is indeed bounded by $\tilde{O}(n^2(\log q)^2)$ bit operations, because we assume that one operation in $\mathbb{F}$ (and also $\mathbb{F}_p$) takes $\tilde{O}(\log q)$ bit operations. ⧫

Remark. It is easy to see that the output of Algorithm 5.22 in Step 2 will actually have all x_i in the *prime field* $\mathbb{F}_p$. In fact, *all* operations of this execution of Algorithm 5.22 are performed within $\mathbb{F}_p$. Therefore, before Step 1, we could replace $\mathbb{F}$ by $\mathbb{F}_p$, and n by $\gcd(n, p-1)$, as described in Section 2.2, to get a better running time bound of $\tilde{O}(n^2(\log p)^2)$ bit operations.

Proof of Theorem 5.3. Algorithm 5.29 claims to solve the problem posed in the Theorem, with the remark that if n does not divide $q-1$, we replace n by $\gcd(n, q-1)$ as described in Section 2.2. By Proposition 5.30, its output satisfies the bounds required by the Theorem and its running time is $\tilde{O}(n^2(\log q)^2)$ bit operations. ⧫

Chapter 6

Diagonal forms

6.1 Introduction and results

We can now complete our deterministic solution of Problems A1 and A2. We have the following results.

Theorem 6.1 *There exists a deterministic algorithm which, given a finite field $\mathbb{F}$ with q elements, a positive integer n, and nonzero elements $a_0, \ldots, a_n$ of $\mathbb{F}$, computes elements $x_0, \ldots, x_n$ of $\mathbb{F}$, not all zero, such that*

$$\sum_{i=0}^{n} a_i x_i^n = 0, \tag{6.2}$$

in time polynomial in n and $\log q$.

An explicit solution method for this homogeneous equation is given below (Section 2, Algorithm 6.7). It assumes the ability to find a sum of nth powers in $\mathbb{F}$ that evaluates to zero, as discussed in Section 5.5. The method maintains a system of equations of trapezoid form, hence the name of *Trapezium algorithm.*

Theorem 6.3 *There exists a deterministic algorithm which, given a finite field $\mathbb{F}$ with q elements, a positive integer n, and nonzero elements $a_1, \ldots, a_n$ and b of $\mathbb{F}$, computes elements $x_1, \ldots, x_n$ of $\mathbb{F}$ such that*

$$\sum_{i=1}^{n} a_i x_i^n = b, \tag{6.4}$$

or correctly asserts that no such elements exist, in time polynomial in n and $\log q$.

The algorithm for this inhomogeneous problem (Algorithm 6.11 in Section 3) consists in a modification of the homogeneous method, this time assuming the ability to write a given element of $\mathbb{F}$ as a sum of nth powers (Section 5.2).

Note that there are no assumptions whatsoever on the field $\mathbb{F}$ or its characteristic.

6.2 The homogeneous trapezium algorithm

We prove Theorem 6.1; let $\mathbb{F}$ be a finite field of q elements, and let n be a positive integer dividing $q-1$. The assumption that n divide the size of the multiplicative group is not a restriction (cf. Section 2.2), but serves only to simplify the complexity estimates. We are given a diagonal form

$$f = a_0 X_0^n + a_1 X_1^n + \ldots + a_n X_n^n,$$

and we exhibit an algorithm for finding a nontrivial zero of this form.

Initialisation. The algorithm starts by computing elements $y_0, \ldots, y_m \in \mathbb{F}$, with $y_0 \neq 0$ and $m \leq n$, whose nth powers sum to zero. This task is completed efficiently and deterministically by Algorithm 5.29, given in Section 5.5. We remove redundancy in the y_i by checking that $y_1^n + \ldots + y_k^n \neq 0$ for $k = 1, \ldots, m$, and discarding $y_1, \ldots, y_k$ for k as large as possible if the test fails.

Once computed, the y_i obviously satisfy the following equations:

$$\begin{cases} a_0 y_0^n = -a_0(y_1^n + \ldots + y_m^n) \\ a_1 y_0^n = -a_1(y_1^n + \ldots + y_m^n) \\ \quad \vdots \qquad\qquad \vdots \\ a_n y_0^n = -a_n(y_1^n + \ldots + y_m^n) \end{cases} \tag{6.5}$$

Data. The algorithm maintains at all times a system of equations of the form

$$\begin{cases} a_0 x_{0,0}^n = -a_0(y_1^n + \ldots + y_{m_0}^n) \\ a_0 x_{1,0}^n + a_1 x_{1,1}^n = -a_1(y_1^n + \ldots + y_{m_1}^n) \\ \quad \iddots \qquad \iddots \qquad \vdots \qquad\qquad \vdots \\ a_0 x_{n,0}^n + a_1 x_{n,1}^n + \ldots + a_n x_{n,n}^n = -a_n(y_1^n + \ldots + y_{m_n}^n), \end{cases} \tag{6.6}$$

where the m_i are integers satisfying $0 \leq m_i \leq m$, the y_i are the same as in (6.5), and the $x_{i,j}$ are in $\mathbb{F}$, and for each i, at least one $x_{i,j}$ is nonzero. Because system (6.6) has a trapezoid form, the current algorithm is called the "trapezium algorithm".

The initial values for (6.6) are given by the system (6.5), with $x_{i,i} = y_0$ for all i, and $x_{i,j} = 0$ when $0 \leq j < i$; every round of the algorithm, in a way to be described shortly, decreases one of the m_i. As soon as one of the m_i becomes zero, the corresponding equation describes a nontrivial representation of zero by the form f, and we are done.

Reduction step. The reduction of the m_i is done as follows. Write S_i for the left hand sides of (6.6), so

$$S_i = a_0 x_{i,0}^n + a_1 x_{i,1}^n + \ldots + a_i x_{i,i}^n \qquad (i = 0, \ldots, n).$$

If any of these happens to be zero, then this also means that $y_1^n + \ldots + y_{m_i}^n$ is zero, and we already removed this kind of redundancy before.

As therefore all S_i are nonzero, we apply Selective Root Extraction (Algorithm 3.12) to them, and find integers k and l, with $k < l$, and $\beta \in \mathbb{F}^*$, such that

$$S_l = \beta^n S_k.$$

This means that we may replace the left hand side of equation l in (6.6) by the left hand side of equation k multiplied by β^n. Equation l thus becomes

$$a_0(\beta x_{k,0})^n + \ldots + a_k(\beta x_{k,k})^n + a_{k+1}0^n + \ldots + a_l 0^n = -a_l(y_1^n + \ldots + y_{m_l}^n),$$

and we see that the last term on the right may be moved to the left without destroying the form of the equation. In other words, m_l is decreased by 1. Note also that the new $x_{l,j}$ are not all zero.

It follows that after at most n^2 steps one of the m_i will become zero, and the algorithm is finished. Here we use the fact that initially we have $m_i \le n$ for all i, and also that m_0 remains unchanged throughout the algorithm.

Algorithm 6.7 (Trapezium; homogeneous case.)
Input: a finite field $\mathbb{F}$ having q elements, a positive integer n dividing $q-1$, and elements $a_0, \ldots, a_n \in \mathbb{F}^*$.
Output: elements $(x_i)_{i=0}^n$ of $\mathbb{F}$, not all zero, such that $\sum_{i=0}^n a_i x_i^n = 0$.

```
1: [Compute zero sequence] Using Algorithm 5.29, compute a sequence $(y_i)_{i=0}^m$ of
   elements of $\mathbb{F}^*$, with $y_0 \neq 0$, such that $\sum_{i=0}^m y_i^n = 0$.
2: [Remove redundancy] Let $k$ be maximal with $0 \le k \le m-1$ such that $\sum_{i=1}^k y_i^n =$
   $0$; discard $y_1, \ldots, y_k$, renumber the remaining $y_i$, and replace $m$ by $m-k$. [When
   $k = 0$, nothing happens.]
3: [Initialise trapezium] For $i = 0, \ldots, n$:
     a: Put $m_i = m$, $x_{i,i} = y_0$, and $S_i = a_i y_0^n$. For $j = 0, \ldots, i-1$, put $x_{i,j} = 0$.
4: [Finished?] While $m_i > 0$ for all $i = 0, \ldots, n$:
     a: [Compare left hand sides] Using Algorithm 3.12, find integers $k$ and $l$ and
        an element $\beta \in \mathbb{F}^*$ such that $0 \le k < l \le n$ and $S_l = \beta^n S_k$.
     b: [Replace big by small] For $j = 0, \ldots, k$, replace $x_{l,j}$ by $\beta x_{k,j}$. For $j =$
        $k+1, \ldots, l-1$, replace $x_{l,j}$ by 0.
     c: [Move term to left] Replace $x_{l,l}$ by $y_{m_l}$, replace $S_l$ by $S_l + a_l y_{m_l}^n$, and replace
        $m_l$ by $m_l - 1$.
5: [Result] Let $i$ be such that $m_i = 0$. Output $(x_{i,j})_{j=0}^i$, followed by $(0)_{j=i+1}^{n+1}$.
```

Proposition 6.8 *Algorithm 6.7 is correct and deterministic, and finishes using* $\tilde{O}(n^3(\log q)^2 + n^2(\log q)^3)$ *bit operations, where* $q = |\mathbb{F}|$.

Proof. After Step 2, the sequence $(y_i)_{i=0}^m$ satisfies $\sum_{i=0}^m y_i^n = 0$, and $\sum_{i=1}^k y_i^n \neq 0$ for $k = 1, \ldots, m$.

Step 3 computes the initial values of the variables of the system (6.6) and also the left hand sides S_j (for $0 \leq j \leq n$). Initially, we have $S_j \neq 0$ for all j because y_0 and the coefficients a_i are nonzero.

From the discussion above, it follows easily that the system (6.6) holds whenever the algorithm enters the loop in Step 4, and also after Step 4 is finished. This includes the condition that for each i (with $0 \leq i \leq n$) not all the $x_{i,j}$ are zero.

Only one of the equations in (6.6) is changed in every execution of Step 4, and it is equation l. Step 4c makes these changes.

The loop in Step 4 will terminate because one of the m_i is decreased during every execution of it. If we have $m_i = 0$ for some i, then we are finished.

Let us bound the running time of this algorithm; write $q = |\mathbb{F}|$. Step 1 takes $\tilde{O}(n^2(\log q)^2)$ bit operations by Proposition 5.30. As we have $m \leq n$ by the same Proposition, and since m_0 is never changed, it follows that Step 4 is executed at most n^2 times. Hence by Proposition 3.13, Step 4 takes $\tilde{O}\left(n^2 \cdot (n(\log q) + (\log q)^2)\right)$ operations in $\mathbb{F}$, which is $\tilde{O}(n^3(\log q)^2 + n^2(\log q)^3)$ bit operations. ♦

Remark. If the prime factors of the exponent n occur only to a low order in the multiplicative group order $q-1$ of $\mathbb{F}^*$, then we have a better bound for the running time (see also the remarks after Proposition 3.13). Namely, if $v_\ell(q-1) = O(\sqrt{\log q / \log \ell})$ for all primes $\ell \mid n$, then by Lemma 3.14 we find a bound of $\tilde{O}(n^3(\log q)^2)$ for Step 4, and thus for the entire Algorithm 6.7.

Proof of Theorem 6.1. If n does not divide $q-1$, we first replace n by $\gcd(n, q-1)$ as described in Section 2.2. Now Algorithm 6.7 claims to solve the homogeneous diagonal equation (6.2). By Proposition 6.8, it is correct and deterministic, and its running time is polynomial in n and $\log q$. This proves Theorem 6.1. ♦

6.3 The inhomogeneous trapezium algorithm

We now turn to the problem of solving (6.4), which is the corresponding inhomogeneous representation problem. This problem is soluble whenever $q > n^2$, and sometimes insoluble otherwise. As regards the first of these two cases, we show that with some small adaptations, we can use the trapezium method for the inhomogeneous problem as well. In the second case, where n is large in comparison with q, we show that an exhaustive search for representations of b is possible in polynomial time.

The adaptations. The first observation is that to solve (6.4), it is enough to solve

$$a_0 x_0^n + a_1 x_1^n + \ldots + a_n x_n^n = 0, \tag{6.9}$$

with $a_0 = -b$ and x_0 a new variable, provided the computed solution satisfies $x_0 \neq 0$.

The second is that, if we have $x_{i,0} \neq 0$ for all i in (6.6) before entering Step 4 of Algorithm 6.7, then this property still holds after Step 4, and thus also when the algorithm finishes; this indeed yields a solution of (6.9) with $x_0 \neq 0$.

Initialisation. We see that it is enough to find an initialisation of the system (6.6) which has $x_{i,0} \neq 0$ for $i = 0, \dots, n$. If every element of $\mathbb{F}$ can be written as a sum of nth powers in $\mathbb{F}$, this is accomplished as follows.

Using Algorithm 5.22 repeatedly, compute elements $y_{i,j} \in \mathbb{F}$ such that

$$b/a_i = \sum_{j=1}^{m_i} y_{i,j}^n \quad (i = 0, \dots, n),$$

where we write $a_0 = -b$ as before. Then the following equations are satisfied:

$$\begin{cases} a_0 = -a_0(y_{0,1}^n + \dots + y_{0,m_0-1}^n) \\ a_0 + a_1 y_{1,m_1}^n = -a_1(y_{1,1}^n + \dots + y_{1,m_1-1}^n) \\ \quad \vdots \qquad\qquad \vdots \\ a_0 + a_n y_{n,m_n}^n = -a_n(y_{n,1}^n + \dots + y_{n,m_n-1}^n) \end{cases}, \tag{6.10}$$

where we have $1 \leq m_i \leq n$ for $i = 0, \dots, n$. However, the $y_{0,j}$ are never used, so in the final algorithm we do not compute them at all, and the topmost equation is just

$$a_0 = a_0.$$

We cannot leave this equation out, as we need $n + 1$ equations in all. It follows that initially in (6.6) we have $x_{i,0} = 1$ for all i, and the same reduction procedure as in the homogeneous case will produce a solution to (6.9) having $x_0 \neq 0$, as desired.

The case where $n^2 > q$. Now Algorithm 5.22 could return "no solution" for some i, which implies that b/a_i is not a sum of nth powers in $\mathbb{F}$. This entails that we are unable to find a solution to (6.4) by the methods just discussed, while still it is possible for one to exist.

By Proposition 2.2, this situation can only occur if $q < n^2$. Now if n is that large, we can use a dynamic programming approach to enumerate all values represented by the form $\sum_{i=1}^n a_i X_i^n$, until all are exhausted or we encounter a solution to (6.4) — see the proof of Proposition 6.12 below for more details. Therefore, in this case we skip Algorithm 5.22 altogether.

It follows that all calls to Algorithm 5.22 done in our algorithm will succeed, and that we can trust our algorithm to find a solution to (6.4) whenever one exists.

Algorithm 6.11 (Trapezium; inhomogeneous case.)
Input: a finite field $\mathbb{F}$ having q elements, a positive integer n dividing $q - 1$, and elements $a_1, \dots, a_n$, and $b \in \mathbb{F}^*$.
Output: elements $(x_i)_{i=1}^n$ of $\mathbb{F}$ such that $\sum_{i=1}^n a_i x_i^n = b$, or, if no such elements exist, "no solution".

```
1: [Large $n$?] If $q < n^2$, enumerate the values taken on by the form $\sum_{i=1}^n a_i X_i^n$ on all possible vectors $(x_1, \ldots, x_n)$ by means of dynamic programming, until they are exhausted or the value $b$ is found. Output "no solution", c.q. the found representation of $b$, and terminate. [See the proof of Proposition 6.12 below for more details.]
2: [Equation 0 is easy] Put $a_0 = -b$. Put $x_{0,0} = 1$, $m_0 = 1$, and $S_0 = a_0$.
3: [Initialisation of rest] For $i = 1, \ldots, n$:
    a: [Sum of powers for $b/a_i$] Using Algorithm 5.22, compute a sequence $(y_{i,j})_{j=1}^{m_i}$ of elements of $\mathbb{F}^*$ with $1 \le m_i \le n$, such that $\sum_{j=1}^{m_i} y_{i,j}^n = b/a_i$.
    b: [Remove redundancy] Let $k$ be maximal such that $\sum_{j=1}^{k} y_{i,j}^n = 0$; discard $y_{i,1}$ up to $y_{i,k}$, renumber the remaining $y_{i,j}$, and replace $m_i$ by $m_i - k$.
    c: Put $x_{i,0} = 1$, $x_{i,i} = y_{i,m_i-1}$ and $S_i = a_0 + a_i x_{i,i}^n$ and replace $m_i$ by $m_i - 1$. For $j = 1, \ldots, i-1$, put $x_{i,j} = 0$.
4: [Finished?] While $m_i > 0$ for all $i = 0, \ldots, n$:
    a: [Compare left hand sides] Using Algorithm 3.12, find integers $k$ and $l$ and an element $\beta \in \mathbb{F}^*$ such that $0 \le k < l \le n$ and $S_l = \beta^n S_k$.
    b: [Replace big by small] For $j = 0, \ldots, k$, replace $x_{l,j}$ by $\beta x_{k,j}$. For $j = k+1, \ldots, l-1$, replace $x_{l,j}$ by 0.
    c: [Move term to left] Replace $x_{l,l}$ by $y_{l,m_l}$, replace $S_l$ by $S_l + a_l y_{l,m_l}^n$, and replace $m_l$ by $m_l - 1$.
5: [Result] Let $i$ be such that $m_i = 0$. Output $(x_{i,j}/x_{i,0})_{j=1}^i$, followed by $(0)_{j=i+1}^n$.
```

Proposition 6.12 *Algorithm 6.11 is correct and deterministic. It uses*

$$\tilde{O}(n^3(\log q)^2 + n^2(\log q)^3)$$

operations in $\mathbb{F}_q$ to finish if $q > n^2$, and $O(n^4 \log q)$ bit operations if $q < n^2$.

Proof. Write $f = \sum_{i=0}^n a_i X_i^n$. The correctness of Algorithm 6.11 follows from the discussion above. We only note that in Step 3b, we make sure that the sequences $(y_{i,j})$, for $i = 0, \ldots, n$, do not have initial subsequences whose nth powers sum to zero. Therefore, the elements S_i are always nonzero.

The loop in Step 4 will terminate because one of the m_i is decreased during every execution of it, and again m_0 is unchanged. In fact, at most $\sum_{i \ge 1}(m_i - 1)$ iterations are performed, which is at most $n(n-1)$ by Proposition 5.22.

It remains to bound the complexity of the algorithm. In Step 1, we keep a table of all elements of $\mathbb{F}$ that we can represent by subforms of the form $f = \sum_{i=1}^n a_i X_i^n$. Initially, the table is empty. In the kth round, we compute all sums of an element in the table and an element of the form $a_k x^n$, with $x \in \mathbb{F}$, and store them in the table, unless they are already there. This is continued until all representations by f are exhausted, or until b is found.

Step 1 takes $O(n^2 \log n + n^4)$ operations in $\mathbb{F}$ for raising n^2 elements to the power n and performing n dynamic programming rounds, where every round considers at most $n \times n^2$ pairs of elements. In total, this gives $O(n^4 \log q)$ *bit* operations.

By Proposition 5.23, Step 3 takes $\tilde{O}(n^3(\log q)^2 + n^2(\log q)^3)$ bit operations for n calls to Algorithm 5.22. Finally, as in the case of the homogeneous algorithm, Step 4 takes $\tilde{O}(n^3(\log q)^2 + n^2(\log q)^3)$ bit operations. ♦

Proof of Theorem 6.3. If n does not divide $q-1$, we first replace n by $\gcd(n, q-1)$ as described in Section 2.2. Now Algorithm 6.11 claims to solve the inhomogeneous equation (6.4). By Proposition 6.12, it is correct and deterministic, and its running time is polynomial in n and $\log q$. This proves Theorem 6.3. ♦

Remark. Our algorithms for solving Problems A1 and A2 are closely related to the proof of Theorem 2.3 and, in particular, to that of Proposition 2.4. In fact, the left hand sides of the trapezium system (6.6) correspond to the subforms f_k of f used in Proposition 2.4, while the equations in the system show certain elements being both written as a sum of nth powers (times a_k) and represented by one of these subforms.

One is led to think that the proof and the algorithm are in a way isomorphic, and it could be interesting to give the precise connection between them.

Chapter 7

Conclusions, generalisations, and applications

7.1 Introduction

In the previous chapters, we have shown that it is easy to compute a nontrivial zero of a diagonal form in sufficiently many variables over a finite field, and also to compute a representation of a given nonzero element by such a form. This gives a complete algorithmic solution of Problems A1 and A2 as given in the Abstract.

We now broaden the perspective and describe the consequences of this result. We begin by considering the performance of our methods when compared to probabilistic methods.

After that, we list several applications to deterministic and efficient computation over finite fields. We show how to compute a generator of prescribed norm for a given extension of finite fields, and we give a complementary approach to the results of Chapter 6 for finite fields of characteristic 2; these applications only use results from Chapters 3 and 4. The others use the main theorem of Chapter 6; they are restricted to case of quadratic forms, as here every form can be brought into diagonal shape. This leaves some interesting questions, such as whether it is easy to compute deterministically a divisor of zero in a central simple algebra of degree at least 3. Finally, it is shown how a solution method for quadratic forms can be used in the deterministic computation of rational points on elliptic curves.

7.2 A performance comparison

Previously, only probabilistic algorithms were available for the task of solving Problems A1 and A2 efficiently. We have given such an algorithm in Section 2.6. It is interesting to ask if the deterministic methods of this thesis can outperform this probabilistic method, at least in some cases.

For this, we compare Proposition 2.12 with, respectively, Propositions 5.30, 5.23, 6.8, and 6.12. I have strived to minimise the asymptotic complexity of all algorithms, and to provide optimal running time bounds for them; and for a fair comparison, I did the same for the probabilistic algorithm analysed in Proposition 2.12.

Proposition 2.12 claims that a probabilistic problem can solve the problems given above in expected time $\tilde{O}(n^2 + n \log q)$ operations in $\mathbb{F}_q$, where n is the degree of the equation, provided we have $q \gg n^2$. If $q \ll n^2$, there may be too few solutions to enable a probabilistic method to work.

One should note that the following complexity bounds are proved assuming the use of fast integer and polynomial arithmetic (see [22] for exact bounds and proofs).

(i) For the problem of computing a sequence of nth powers that sum to zero, the deterministic algorithm 5.29 runs equally fast as the probabilistic method for fixed n, as its execution takes $\tilde{O}(n^2(\log q)^2)$ bit operations.

(ii) For the problem of representing a given nonzero element as a sum of nth powers, the deterministic algorithm 5.22 takes more time than the probabilistic method, except in the cases where $q \ll n^2$, where the probabilistic method does not work. Its execution takes $\tilde{O}(n^2(\log q)^2 + n(\log q)^3)$ bit operations.

(iii) For finding a nontrivial zero of a diagonal form with arbitrary coefficients, the deterministic algorithm 6.7 takes $\tilde{O}(n^3(\log q)^2 + n^2(\log q)^3)$ bit operations. This is more than the probabilistic method, but the method works also if $q \ll n^2$, in which case the latter breaks down.

(iv) For finding a representation of a nonzero element by a diagonal form with arbitrary coefficients, the deterministic algorithm 6.11 takes also $\tilde{O}(n^3(\log q)^2 + n^2(\log q)^3)$ bit operations, except in the case where $q < n^2$, where it may not work at all. In this case, where the probabilistic method breaks down as well, I know no other method but enumerating all possibilities; if this is done sensibly, by means of a dynamic programming approach, it takes $O(n^4 \log q)$ bit operations.

One sees that the running time of all these deterministic algorithms, except for the first, is cubic in terms of $\log q$. As I described in detail in Sections 3.4 and 5.5, this could improve to quadratic if a faster root taking method could be used in our algorithms.

7.3 Field generators of prescribed norm

The next application deals with finding generators for extensions of finite fields with special properties. Let $\mathbb{E} \subseteq \mathbb{F}$, of degree e, be such an extension; it is easy to construct a generator b of $\mathbb{F}$ over $\mathbb{E}$ such that, for example, the *trace* of b in the extension $\mathbb{F}/\mathbb{E}$ is equal to some given element $a \in \mathbb{E}$ (except that when the characteristic and the extension degree are both 2, a generator cannot have trace 0). This says, in fact, that we can usually prescribe the coefficient of X^{e-1} in the minimal polynomial of a generator b. Also, as will be shown in the present section, it is possible to require the *norm* of a generating element to take any nonzero value; this corresponds to prescribing the constant term of the minimal polynomial.

The question of whether it is possible to prescribe an arbitrary coefficient is known as the *Hansen-Mullen conjecture* [27, 54]. Another version of this conjecture restricts attention to *primitive elements*, i.e., field generators for $\mathbb{F}/\mathbb{E}$ that also generate the multiplicative group $\mathbb{F}^*$. For a recent overview of this problem, we refer to [17].

The Hansen-Mullen conjecture has been settled, except possibly for finitely many combinations of $\mathbb{E}$ and e, and even some extensions have been proved, like cases where it is possible to prescribe *more than one* coefficients in the minimal polynomial of a primitive element. The methods of proof, however, are mostly non-constructive, and it is hence still interesting to construct *algorithms* that determine field generators with desirable properties.

Unfortunately, up to now no efficient method is known for computing primitive elements, let alone to prescribe coefficients of their minimal polynomials. In fact, it is not known if we can test an element for being primitive efficiently, as all known algorithms for this require the factorisation of the order of the multiplicative group $\mathbb{F}^*$.

However, it is possible to give algorithms that construct field generators with prescribed norm, for example. This is easily done probabilistically, if the field is not too small; but the following theorem gives a *deterministic* algorithm for this problem, which works over finite fields of any size. For the proof, we use several of the algorithms that have been developed in this thesis.

Theorem 7.1 *There exists an efficient deterministic algorithm that, given an extension of finite fields $\mathbb{E} \subseteq \mathbb{F}$ and an element $a \in \mathbb{E}^*$, computes $b \in \mathbb{F}$ such that*

(i) *b generates $\mathbb{F}$ over $\mathbb{E}$;*

(ii) $\mathrm{Norm}_{\mathbb{F}/\mathbb{E}}(b) = a$.

Another way to put this result is the following.

Corollary 7.2 *There exists an efficient deterministic algorithm that, given a finite field $\mathbb{E}$, an irreducible polynomial f of degree e with coefficients in $\mathbb{E}$, and an element $a \in \mathbb{E}^*$, computes a monic irreducible polynomial of degree e with coefficients in $\mathbb{E}$ whose constant coefficient is equal to a.*

Proof. Applying the algorithm from the Theorem to $\mathbb{F} = \mathbb{E}[X]/(f)$, compute $b \in \mathbb{F}$ such that Norm $b = (-1)^{e-1}a$. Then the minimal polynomial of b over $\mathbb{E}$ has the required properties. ♦

The proof of Theorem 7.1 is done in three steps, given by the following three Propositions. The first shows how to compute a field generator of norm 1.

Proposition 7.3 *There exists an efficient deterministic algorithm that, given an extension of finite fields $\mathbb{E} \subseteq \mathbb{F}$ of degree e, computes a generator c for $\mathbb{F}$ over $\mathbb{E}$ such that* $\operatorname{Norm}_{\mathbb{F}/\mathbb{E}}(c) = 1$.

Proof. We consider the subfield K consisting of all sums of eth powers in $\mathbb{F}$. By Proposition 2.2, we have $K = \mathbb{F}$ whenever $e^2 < |\mathbb{F}|$. But if $e^2 \geq |\mathbb{F}|$, then it follows that $\mathbb{E} = \mathbb{F}_2$ and $e = 2, 3, 4$, and in these cases e is relatively prime to $|\mathbb{F}| - 1$, so that every element of $\mathbb{F}$ is an eth power, and again $K = \mathbb{F}$.

Use Algorithm 4.14 to compute a generator α^e for $K = \mathbb{F}$ over $\mathbb{E}$. Then the element

$$c = \frac{\alpha^e}{\operatorname{Norm} \alpha}$$

still generates $\mathbb{F}$ over $\mathbb{E}$, whereas clearly its norm is 1. ♦

Next, we compute an element b in $\mathbb{F}$ (not necessarily a generator) of norm a, with the additional property that $\operatorname{ord} b$ contains only primes that already divide $\operatorname{ord} a$.

Proposition 7.4 *There exists an efficient deterministic algorithm that, given an extension of finite fields $\mathbb{E} \subseteq \mathbb{F}$ of degree e, and an element $a \in \mathbb{E}$, computes $b \in \mathbb{F}$ such that* $\operatorname{Norm}_{\mathbb{F}/\mathbb{E}} b = a$ *and such that* $\operatorname{ord} b$ *has the same prime factors as* $\operatorname{ord} a$.

The proof of this Proposition will use four Lemmata.

Lemma 7.5 *Let $\mathbb{E} \subseteq \mathbb{F}$ be an extension of finite fields of prime degree ℓ. Let a be an element of $\mathbb{E}$. If $b \in \mathbb{F}$ is such that $b^\ell = a$, then*

$$\operatorname{Norm}_{\mathbb{F}/\mathbb{E}}(b) = \begin{cases} a & \textit{if } b \in \mathbb{E}, \textit{ and} \\ (-1)^{\ell-1}\, a & \textit{otherwise.} \end{cases}$$

Proof. If $b \in \mathbb{E}$, then $\operatorname{Norm} b = b^\ell = a$. Otherwise, b generates $\mathbb{F}$ over $\mathbb{E}$, so its minimal polynomial is $X^\ell - a$, and its norm is $(-1)^{\ell-1}\, a$. ♦

Lemma 7.6 *There exists an efficient deterministic algorithm that, given an extension of finite fields $\mathbb{E} \subseteq \mathbb{F}$ of prime degree ℓ, and an element $a \in \mathbb{E}^*$, computes $b \in \mathbb{F}^*$ such that $b^\ell = a$, and such that* $\operatorname{ord} b$ *has the same prime factors as* $\operatorname{ord} a$.

Proof. There are three cases. Note that we do not assume that ℓ is odd.

If ℓ does not divide $|\mathbb{E}| - 1$, we simply write a as the ℓth power of some element b using Algorithm 2.1. The orders of b and a are equal.

If ℓ divides $|\mathbb{E}| - 1$ but not $\operatorname{ord}(a)$, then the first case of Proposition 3.6 shows that an ℓth root b of a exists such that $v_\ell(\operatorname{ord} b) = 0$. It follows that calling Algorithm 3.7 with finite field $\mathbb{E}$, prime ℓ, exponent $f = 1$, and generator $g = 1$ will return such an ℓth root b, and that the orders of b and a will be equal.

Finally, if ℓ divides $\operatorname{ord}(a)$, we have to do some work. Note that by our assumptions, we know that ℓ divides $|\mathbb{E}|-1$ and is different from char $\mathbb{E}$. First, using Lemma 5.25 with $n=1$, we know that there exists an ℓth root of unity in $\mathbb{E}$, and we construct such an element. Then, we use Lagrange resolvents to write $\mathbb{F}$ as a radical extension of $\mathbb{E}$, following the proofs of Theorems VI.6.1 and VI.6.2(ii) of [33]. The Lagrange resolvent is a nonzero $\mathbb{E}$-linear map ϕ on $\mathbb{F}$, such that whenever $\phi(x) \neq 0$ for $x \in \mathbb{F}$, then $\phi(x)$ is such that $\phi(x)^\ell \in \mathbb{E}$ while $\phi(x) \notin \mathbb{E}$. Thus, we must find an element x outside the kernel of ϕ; but ϕ, being linear, must take a nonzero value on at least one element of any basis for $\mathbb{F}$ over $\mathbb{E}$, and this shows that we can construct such an x efficiently and deterministically.

Assume now, as we may, that $\mathbb{F}$ is given as a radical extension of $\mathbb{E}$; that is, we are given an element $c \in \mathbb{F}$ such that $c^\ell \in \mathbb{E}$ while $c \notin \mathbb{E}$. Because c^ℓ is not an ℓth power in $\mathbb{E}$, we see that $\operatorname{ord}(c)$ has more factors ℓ than the order of any element in $\mathbb{E}$. Thus, we can use c as a generator in Algorithm 3.7 to compute an ℓth root in $\mathbb{F}$ of any element in $\mathbb{E}$, and in particular an ℓth root b of a. Finally, it is clear that $\operatorname{ord}(b)/\operatorname{ord}(a)=\ell$, so that $\operatorname{ord}(b)$ and $\operatorname{ord}(a)$ have the same prime factors. ♦

Lemma 7.7 *The task of taking square roots in finite fields is efficiently and deterministically reducible to the task of taking square roots in finite prime fields.*

Proof. Let $\mathbb{F}$ be a finite field of characteristic p, and let $a \in \mathbb{F}$ be a square. Clearly $\operatorname{Norm}_{\mathbb{F}/\mathbb{F}_p}(a)$ is a square in $\mathbb{F}_p$; therefore,

$$\operatorname{Norm}(a^{(p-1)/2}) = (\operatorname{Norm} a)^{(p-1)/2} = 1.$$

Define $A = a^{(p-1)/2}$; by Hilbert's Theorem 90 [33, Theorem VI.6.1], there exists $c \in \mathbb{F}$ such that $c^{p-1} = A$. To find such a c, we solve the equation $c^p = Ac$; the operators $c \mapsto c^p$ and $c \mapsto Ac$ are both $\mathbb{F}_p$-linear, so c can be found by taking any nonzero solution of a linear system of equations over $\mathbb{F}_p$. Alternatively, we can again use the linear operator given in the proof of Theorem VI.6.1 in [33], taking care to select an element of $\mathbb{F}$ where the operator takes a nonzero value.

Now the element c^2/a of $\mathbb{F}$ satisfies

$$\left(c^2/a\right)^{(p-1)/2} = 1;$$

therefore, it is in the prime field, and it is even a square there. Now let d be a square root of c^2/a, given to us by an oracle that computes square roots in $\mathbb{F}_p$. Then clearly c/d is a square root of a, and we are done. ♦

Remark. The same proof shows that, for any prime ℓ and any finite field $\mathbb{F}$, taking ℓth roots in $\mathbb{F}$ is reducible to taking ℓth roots in the smallest subfield of $\mathbb{F}$ that contains an ℓth root of unity.

Lemma 7.8 *There exists an efficient deterministic algorithm that, given an extension of finite fields $\mathbb{E} \subseteq \mathbb{F}$ of prime degree ℓ, computes $b \in \mathbb{F}$ such that $\operatorname{Norm}_{\mathbb{F}/\mathbb{E}} b = -1$ and such that $\operatorname{ord}(b)$ is a power of 2.*

Proof. If char $\mathbb{E} = 2$, we take $b = 1$; if $\ell \neq 2$, we take $b = -1$. Consider the remaining case, viz., $\mathbb{F}$ is quadratic over $\mathbb{E}$ and char $\mathbb{E}$ is odd.

This being so, we compute $c \in \mathbb{F}$ such that $c^2 \in \mathbb{E}$, but $c \notin \mathbb{E}$, as in the proof of Lemma 7.6, using the primitive 2nd root of unity -1. Alternatively, we can take for c the discriminant of the minimal polynomial of any generator for $\mathbb{F}$ over $\mathbb{E}$. Then, again, we can use c as a generator in Algorithm 3.7 to compute a square root in $\mathbb{F}$ of any element in $\mathbb{E}$, and, by Lemma 7.7, even of any square element in $\mathbb{F}$.

Now the norm map projects the 2-Sylow subgroup of $\mathbb{F}^*$ onto that of $\mathbb{E}^*$. Therefore, if we start with $-1 \in \mathbb{E}$ and repeatedly take a square root, we will find an element b of norm -1, such that $\operatorname{ord}(b)$ is a power of 2, as desired. ♦

Proof of Proposition 7.4. We use induction on the prime divisors of e, taken with multiplicities. If $e = 1$, we take $b = a$.

Assume $e > 1$, and let ℓ be a prime divisor of e. Let M be the unique degree ℓ extension of $\mathbb{E}$ contained in $\mathbb{F}$; a generator for M over $\mathbb{E}$ can be computed efficiently and deterministically. We first prove the Proposition for the extension $M/\mathbb{E}$.

Using Lemmata 7.5 and 7.6, compute $b' \in M$ such that $\mathrm{Norm}_{M/\mathbb{E}}\, b' = \pm a$ and such that $\operatorname{ord}(b')$ and $\operatorname{ord}(a)$ have the same prime factors. If $\mathrm{Norm}\, b' = a$, we take $b = b'$, and we are done. If not, we have $\ell = 2$ and char $\mathbb{E} \neq 2$. Furthermore, we have $b \notin \mathbb{E}$, so a is a nonsquare in $\mathbb{E}$, and hence the order of a is even. Now using Lemma 7.8, compute $c \in M$ such that $\mathrm{Norm}\, c = -1$ and $\operatorname{ord}(c)$ is a power of 2. It follows that $\mathrm{Norm}(c\, b') = a$, and $\operatorname{ord}(c\, b')$ has the same prime factors as $\operatorname{ord}(a)$. We take $b = c\, b'$, and we are done.

Now by induction, we can compute some $d \in \mathbb{F}$ such that $\mathrm{Norm}_{\mathbb{F}/M}(d) = b$, and $\operatorname{ord}(d)$ has the same prime factors as $\operatorname{ord}(b)$. Because $\mathrm{Norm}_{\mathbb{F}/\mathbb{E}} = \mathrm{Norm}_{M/\mathbb{E}} \circ \mathrm{Norm}_{\mathbb{F}/M}$, we see that d satisfies the requirements of the Proposition. ♦

Remark. The approach given above, proceeding by extensions of prime degree only, has been chosen for making the proofs simple. It is also possible to give a more direct approach, which will eliminate the need for computing a complete chain of subfields of $\mathbb{F}$, and will therefore be preferable in case of implementation. This remark pertains especially to Lemmata 7.5 and 7.6.

The third step of the computation is very simple.

Proposition 7.9 *Let $\mathbb{E} \subseteq \mathbb{F}$ be an extension of finite fields, and let $a \in \mathbb{E}^*$. Furthermore, let $b \in \mathbb{F}$ be such that* $\mathrm{Norm}_{\mathbb{F}/\mathbb{E}}\, b = a$ *and such that* $\operatorname{ord}(b)$ *and* $\operatorname{ord}(a)$ *have the same prime factors. Finally, let $c \in \mathbb{F}$ be a generator for $\mathbb{F}$ over $\mathbb{E}$ such that* $\mathrm{Norm}_{\mathbb{F}/\mathbb{E}}\, c = 1$.

Then the element bc has norm a and generates $\mathbb{F}$ over $\mathbb{E}$.

Proof. We obviously have $\mathrm{Norm}(bc) = a$. We claim that, for any prime ℓ dividing $\operatorname{ord}(c)$, we have $v_\ell(\operatorname{ord}(bc)) \geq v_\ell(\operatorname{ord} c)$. Thus, c is contained in the subgroup of $\mathbb{F}^*$ generated by bc, and hence bc is also a field generator for $\mathbb{F}$ over $\mathbb{E}$.

We prove the claim. Let the prime ℓ divide $\mathrm{ord}(c)$. If ℓ does not divide $\mathrm{ord}(b)$, the claim is evident. Now suppose ℓ divides $\mathrm{ord}(b)$. Then ℓ also divides $\mathrm{ord}(a) = \mathrm{ord}(\mathrm{Norm}\, b)$, but not $\mathrm{ord}(\mathrm{Norm}\, c) = 1$. We see that $\mathrm{ord}(b)$ has more factors ℓ than $\mathrm{ord}(c)$, and the claim is proved. ♦

We are now able to give the proof of the main theorem of this section.

Proof of Theorem 7.1. Using the algorithm from Proposition 7.3, compute $c \in \mathbb{F}$ such that $\mathrm{Norm}_{\mathbb{F}/\mathbb{E}}\, c = 1$. Then, using the algorithm from Proposition 7.4, compute $b \in \mathbb{F}$ such that $\mathrm{Norm}\ b = a$ and $\mathrm{ord}(b)$ has the same prime factors as $\mathrm{ord}(a)$. Then the element bc has the desired properties, by Proposition 7.9.

The resulting algorithm is clearly efficient and deterministic. ♦

7.4 Diagonal forms in characteristic 2

This section proves the following statements. Let $\mathbb{F}$ be an extension of $\mathbb{F}_2$ of degree e, and let n be a positive integer. Then every diagonal form of degree n in $e+1$ variables over $\mathbb{F}$ has a nontrivial zero, which can be computed efficiently and deterministically. Furthermore, if the nth powers in $\mathbb{F}$ are not contained in a proper subfield of $\mathbb{F}$, then every diagonal form of degree n in e variables over $\mathbb{F}$ is universal, and representations of nonzero elements of $\mathbb{F}$ by such forms can be computed efficiently and deterministically.

These statements are obvious improvements of Theorems 6.1 and 6.3 in the cases where $n > e$. The first one is very easily proved, while the proof of the second uses the main theorem of Chapter 4.

Theorem 7.10 *There exists an efficient deterministic algorithm that, given an extension $\mathbb{F}/\mathbb{F}_2$ of degree e, a positive integer n, and elements $a_0, \ldots, a_e \in \mathbb{F}^*$, computes $x_0, \ldots, x_e \in \mathbb{F}$, not all zero, such that*

$$a_0 x_0^n + a_1 x_1^n + \ldots + a_e x_e^n = 0,$$

in time polynomial in e and n.

Proof. As $\mathbb{F}$ has degree e, the a_i are linearly dependent over $\mathbb{F}_2$. Therefore, we can take the x_i to be the coefficients of an arbitrary nontrivial $\mathbb{F}_2$-linear combination of the a_i; we will have $x_i = x_i^n$ for all i, and we obtain the desired solution. ♦

Theorem 7.11 *There exists an efficient deterministic algorithm that, given an extension $\mathbb{F}/\mathbb{F}_2$ of degree e, a positive integer n, and elements $a_1, \ldots, a_e$ and b of $\mathbb{F}^*$, computes $x_1, \ldots, x_e \in \mathbb{F}$ such that*

$$a_1 x_1^n + \ldots + a_e x_e^n = b,$$

or shows that no such x_i exist, in time polynomial in e and n.

The proof is based on the following corollary of Theorem 4.2, which is actually valid in any positive characteristic. I thank Arne Winterhof for suggesting this method of proof.

Lemma 7.12 *There exists a deterministic algorithm that, given a finite field $\mathbb{F}$ of extension degree e over its prime field $\mathbb{F}_p$, a positive integer n such that all elements of $\mathbb{F}$ are sums of nth powers in $\mathbb{F}$, and elements $a_1, \ldots, a_e$ in $\mathbb{F}^*$, computes a basis for $\mathbb{F}$ over $\mathbb{F}_p$ of the form*

$$\{a_1 x_1^n, a_2 x_2^n, \ldots, a_e x_e^n\}$$

with $x_1, \ldots, x_e \in \mathbb{F}^*$, in time polynomial in $\log p$, e, and n.

Proof. By our assumption on sums of nth powers in $\mathbb{F}$, we can use Algorithm 4.14 to compute a generator α^n for $\mathbb{F}$ over $\mathbb{F}_p$. It follows that, for $i = 1, \ldots, e$, the set

$$\{a_i(\alpha^0)^n, a_i(\alpha^1)^n, \ldots, a_i(\alpha^{e-1})^n\}$$

is linearly independent over $\mathbb{F}_p$.

We construct the desired basis by induction. Take $a_1(\alpha^0)^n = a_1 \neq 0$ as a first element. Next, take i with $2 \leq i \leq e$, and suppose we have a linearly independent set $\{a_1 x_1^n, \ldots, a_{i-1} x_{i-1}^n\}$. By the above reasoning, there is at least one k such that $a_i(\alpha^k)^n$ is linearly independent from this set, and we can therefore expand our set to have i elements. This completes the construction.

The resulting algorithm is clearly deterministic and takes polynomial time. ♦

Proof of Theorem 7.11. We may replace n by $\gcd(n, 2^e - 1)$ using Algorithm 2.1. After this, if we have $n^2 > 2^e$, we simply check all possibilities, using the same dynamic programming approach as in the proof of Proposition 6.12.

Now assume we have $n^2 < 2^e$, so that $\mathbb{F}$ is equal to its subfield of sums of nth powers. Use the algorithm from the Lemma to compute a basis for $\mathbb{F}$ over $\mathbb{F}_2$ of the form

$$(a_1 y_1^n, \ldots, a_e y_e^n).$$

The element b is an $\mathbb{F}_2$-linear combination of these basis elements. Let $b_i \in \mathbb{F}_2$, for $i = 1, \ldots, e$, be the coefficients of this linear combination; then $(b_i y_i)_{i=1}^e$ is the desired solution. ♦

Remark. It is at present unclear if this approach can be extended to the case of odd characteristic.

7.5 Quadratic forms

We will now list several applications of the algorithms in this thesis. In the present section, we restrict our attention to applications in the special case $n = 2$ — the quadratic case. The reason for this is evident: over a field of odd characteristic, every quadratic form can be *diagonalised* by an efficient deterministic algorithm, so that our methods can in fact be applied to every multivariate polynomial of total degree 2 over a finite field.

In fields of characteristic 2, the operation of squaring is an $\mathbb{F}_2$-linear map, and diagonalisation is no longer possible. This leads to essentially different methods for finding zeros of quadratic polynomials, mainly involving linear algebra, for which the algorithms of this thesis are unnecessary. One result obtained using such methods is that every homogeneous quadratic equation in at least 3 variables can be solved efficiently and deterministically over finite extensions of $\mathbb{F}_2$.

Basic equation. In the special case $n = 2$, we see that solving Problem A2 is the same as solving

$$ax^2 + by^2 = c, \tag{7.13}$$

whenever $abc \neq 0$. It is well known that this is possible; in fact, this follows already from the fact that the cardinalities of the sets $\{ax^2 \mid x \in \mathbb{F}\}$ and $\{c - by^2 \mid y \in \mathbb{F}\}$ add up to more than $|\mathbb{F}|$, for an arbitrary finite field $\mathbb{F}$, and therefore these sets must meet.

As far as I know this thesis gives the first efficient deterministic algorithm for solving (7.13). It may be useful to remark that even for $n = 2$, the complete machinery of Chapters 3 to 6 is required, although it is possible to give a simpler formulation as follows. Upon critical inspection, this formulation will show to be algorithmically equivalent to the trapezium algorithm 6.11.

Thus, let a, b, c be given; we may of course assume that $c = 1$. Now if $v_2(\operatorname{ord} a) > v_2(\operatorname{ord} b)$, we can use Algorithm 3.7 to take a square root of b, and the problem is solved; and analogously if b has the larger order. If $v_2(\operatorname{ord} a) = v_2(\operatorname{ord} b) =_{\text{def}} w$, we distinguish three cases: $w = 0$, $w = 1$, and $w > 1$.

If $w = 0$, then we can still compute square roots of both a and b by means of Algorithm 3.7, and we are done. If $w > 1$, then $v_2(\operatorname{ord}(-ab)) < w$, so that, after computing $\sqrt{-ab}$, we may assume $b = -a$. The equation $ax^2 - ay^2 = 1$ is easily solved by putting $x + y = 1$ and $x - y = 1/a$ and solving the linear system.

The case $w = 1$ is the hardest. Both $-a$ and $-b$ have odd order, so we may take their square roots and obtain the equation $-x^2 - y^2 = 1$. One sees that this is equivalent to

$$x^2 + y^2 + z^2 = 0.$$

For this, we developed a fast algorithm in Section 5.5. A slower, but also deterministic, algorithm for this problem can be found in [14].

Rational points on quadrics. If we can solve equation (7.13) in 2 variables, then obviously we can solve diagonal quadratic equations in more than 2 variables as well. Therefore, we have found an efficient deterministic algorithm that computes a rational point on any projective quadratic hypersurface, of dimension greater than 0, over a finite field of odd characteristic.

Isomorphism of quadratic spaces. The solution of (7.13) is important for several algorithms in the theory of quadratic forms over finite fields of odd characteristic. Notably, we can use the solution method for equation (7.13) repeatedly to establish a deterministic algorithm for computing an embedding of a given quadratic space in another, provided the dimensions are unequal. If the spaces have equal dimension and are isomorphic, then an isomorphism can be computed deterministically, provided we are given a square root of the quotient of the discriminants. These results will be given in detail in a forthcoming publication.

Trivialising central simple algebras. By the Wedderburn theorem [57, Theorem 1], every central simple algebra over a finite field is isomorphic to a matrix algebra over this field. Furthermore, we can actually *compute* an isomorphism, by means of an efficient deterministic algorithm, once we know a divisor of zero in the algebra (see [41]). Such a zero divisor in turn is derived from a nontrivial zero of the *reduced norm form* of the algebra.

The vector space dimension of a central simple algebra over a field is always a square; if this dimension is n^2, the integer n is called the *degree* of the algebra. For further definitions and results about central simple algebras, one could consult [32] or [39].

For algebras of degree 2, the reduced norm form is a quadratic form in four variables. Thus, it follows from the results in this thesis that there exists a deterministic polynomial time algorithm for trivialising central simple algebras of degree 2 over finite fields of odd characteristic.

For degrees higher than 2, the norm form is not diagonalisable. In fact, Corollary 2.8 shows that for a given degree n and over sufficiently large finite fields with respect to n, all diagonal forms in 3 or more variables are isotropic; but a central simple algebra of degree n contains a subfield of extension degree n over the base field, and the norm form of this subalgebra has n variables but no nontrivial zero.

Thus, the question remains open whether an efficient deterministic algorithm exists for constructing a zero divisor in a central simple algebra of degree at least 3 over a finite field.

Solving quadratic equations over the integers (1). One can use Cornacchia's algorithm ([16, Algorithm 1.5.2], [43]) to find a representation of an odd prime p by the principal *binary quadratic form* $X^2 + nY^2$, given a solution of the local equation $x^2 + ny^2 \equiv 0 \pmod{p}$ (here n is a positive integer). This algorithm comes down to a g.c.d. computation over the ring of rational integers.

One notes that no efficient deterministic algorithm is known for solving the congruence $x^2 + ny^2 \equiv 0$, even for a prime modulus: this is equivalent to computing $\sqrt{-n}$ in $\mathbb{F}_p$, for which one has to resort to probabilistic methods.

Using the methods of this thesis, we can in some cases obtain a generalisation of Cornacchia's algorithm for *quaternary* quadratic forms over $\mathbb{Z}$ that is completely deterministic. We only give the example of the classical problem of writing integers as sums of four squares. The form $f = X^2 + Y^2 + Z^2 + W^2$ is the norm form of the ring of *integral quaternions* $\mathcal{A} = \mathbb{Z} + \mathbb{Z}i + \mathbb{Z}j + \mathbb{Z}k$, with the multiplication given by the relations $i^2 = j^2 = k^2 = -1$ and $ij = -ji = k$.

Let p be a prime in $\mathbb{Z}$, and let (x, y, z) be a nontrivial solution of the congruence

$$x^2 + y^2 + z^2 \equiv 0 \pmod{p}. \tag{7.14}$$

Then, although $\mathcal{A}$ is not a principal ideal ring, the right ideal $p\mathcal{A} + (xi + yj + zk)\mathcal{A}$ of norm p is generated by a single element $a + bi + cj + dk$, whose norm is hence equal to p (see Theorems 376–378 of [28]). We can find such a generator by writing $\mathcal{A}$ as $\mathbb{Z}[i] + \mathbb{Z}[i]j$ and applying Cornacchia's algorithm over the base ring $\mathbb{Z}[i]$ instead of $\mathbb{Z}$.

Incidentally, if $p > 2$, then the ring $\mathcal{A}/p$ is a central simple algebra of degree 2 over $\mathbb{F}_p$, and asking for an element of $\mathcal{A}/p$ that has both trace and norm equal to zero gives rise to equation (7.14).

Now this thesis gives an efficient deterministic algorithm to produce a nontrivial solution to (7.14), and it follows that one can write primes as sums of four squares by means of an efficient deterministic algorithm. This application was already given in [14]. If p is allowed to be any positive integer, then we can use a probabilistic algorithm given in [2] to solve (7.14).

It is to be expected that this generalised version of Cornacchia's algorithm will work in all definite quaternion orders whose right ideal class number is 1. Unfortunately, there are only finitely many of these; a complete list is given in [13].

Solving quadratic equations over the integers (2). When using Cornacchia's method to solve $x^2+ny^2 = p$, there is no need to factor the discriminant $4n$. However, if one allows oneself to factor the discriminant of a quadratic form over the integers, then it turns out that a similar combination of global and local methods gives rise to an efficient solution method for forms of rank at least 3. In recent papers, D. Simon has given efficient algorithms for computing zeros of indefinite forms of rank 3 or higher, using a generalisation of the LLL-algorithm to the indefinite case [49, 48]. As the author indicates, the algorithms in these papers can be made deterministic by incorporating the methods of this thesis for finding zeros of quadratic forms over finite fields.

7.6 Rational points on elliptic curves

As a final and surprising application, we mention the task of constructing rational points on an elliptic curve over a finite field. The problem of finding an efficient deterministic algorithm for this task has been open at least since 1985, when Schoof posed it in [42].

Let $\mathbb{F}$ be field of q elements; we first assume that $\gcd(6, q) = 1$. Under these assumptions, an elliptic curve over $\mathbb{F}$ is given by a Weierstrass equation

$$y^2 = f(x) = x^3 + ax + b,$$

where $a, b \in \mathbb{F}$ are such that $f(x)$ has no double roots. The number N of points on E with coordinates in $\mathbb{F}$ (including the point "at infinity") satisfies the *Hasse bound* $|q+1-N| \leq 2\sqrt{q}$. (For these and more results on elliptic curves, I refer to [47].) The question is for an algorithm that computes such a rational point, different from the one at infinity.

From the Hasse bound, one can see that whenever $q > 4$, the curve possesses at least 2 rational points, and already for moderate q, there are sufficiently many rational points on E to make a probabilistic approach feasible. No efficient deterministic approach has been known to date.

However, in a recent paper [50], M. Skałba showed how to compute $x_1, x_2, x_3 \in \mathbb{F}$ such that

$$f(x_1)f(x_2)f(x_3) \in \mathbb{F}^{*2}.$$

From this, it follows that $f(x_i) \in \mathbb{F}^{*2}$ for at least one i, and furthermore, one can apply Algorithm 3.7 to compute the square root, which completes the task.

Skałba's algorithm does not cover the case $j = 0$ (which is equivalent to $b = 0$, using a Weierstrass equation as above). Pursuing his approach, I have found a simpler method that does incorporate this case, and additionally the case of characteristic 3. Interestingly, the ensuing algorithm involves solving an equation of the form $ax^2 + by^2 = c$ over $\mathbb{F}$, which can be done efficiently and deterministically using Algorithm 6.11 of this thesis. We have hence reduced the task of finding a point on a curve of genus 1 to that of finding a point on a curve of genus 0.

The case of characteristic 2 is completely different on the face of it; for example, one has to use a more general form of Weierstrass equation (cf. Appendix A of [47]). However, the same geometric ideas as in the case of odd characteristic suffice to produce an efficient point finding algorithm for this case as well; a salient feature is again a reduction to point finding on a curve of genus 0.

The exact result is as follows; details and proofs will be given in a forthcoming publication [45]. This is a shared publication, as I recently learned that a proof of the case of characteristic 2 was given by A. Shallue a few months before mine.

Theorem 7.15 *There exists a deterministic algorithm that, given a finite field $\mathbb{F}$ and a Weierstrass equation for an elliptic curve E over $\mathbb{F}$, computes a point on E with coordinates in $\mathbb{F}$ other than the point at infinity, in time polynomial in $|\mathbb{F}|$, or proves that no such point exists.*

References

[1] Leonard Adleman, Kenneth Manders, and Gary Miller. On taking roots in finite fields. In *18th Annual Symposium on Foundations of Computer Science (Providence, R.I., 1977)*, pages 175–178. IEEE Comput. Sci., Long Beach, Calif., 1977.

[2] Leonard M. Adleman, Dennis R. Estes, and Kevin S. McCurley. Solving bivariate quadratic congruences in random polynomial time. *Math. Comp.*, 48(177):17–28, 1987.

[3] Shigeki Akiyama, Christiane Frougny, and Jacques Sakarovitch. On the representation of numbers in a rational base. In *5th International Conference on Words*. Publications du LaCIM, Université du Québec à Montréal, September 2005.

[4] Eric Bach. A note on square roots in finite fields. *IEEE Trans. Inform. Theory*, 36(6):1494–1498, 1990.

[5] Eric Bach. Comments on search procedures for primitive roots. *Math. Comp.*, 66(220):1719–1727, 1997.

[6] Eric Bach, Joachim von zur Gathen, and Hendrik W. Lenstra, Jr. Factoring polynomials over special finite fields. *Finite Fields Appl.*, 7(1):5–28, 2001. Dedicated to Professor Chao Ko on the occasion of his 90th birthday.

[7] Eric Bach and Jeffrey Shallit. *Algorithmic number theory. Vol. 1: Efficient algorithms.* Foundations of Computing Series. MIT Press, Cambridge, MA, 1996.

[8] Daniel J. Bernstein. Detecting perfect powers in essentially linear time. *Math. Comp.*, 67(223):1253–1283, 1998.

[9] Daniel J. Bernstein. Faster square roots in annoying finite fields. Draft. URL: `http://cr.yp.to/papers.html#sqroot`, 2005.

[10] Thomas Beth, Dieter Jungnickel, and Hanfried Lenz. *Design theory. Vols. I and II*, volume 69 and 78 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, second edition, 1999.

[11] Olaf Bonorden, Joachim von zur Gathen, Jürgen Gerhard, Olaf Müller, and M. Nöcker. Factoring a binary polynomial of degree over one million. *SIGSAM Bull.*, 35(1):16–18, 2001.

[12] Wieb Bosma, John Cannon, and Catherine Playoust. The Magma algebra system. I. The user language. *J. Symbolic Comput.*, 24(3-4):235–265, 1997. Computational algebra and number theory (London, 1993).

[13] Juliusz Brzezinski. Definite quaternion orders of class number one. *J. Théor. Nombres Bordeaux*, 7(1):93–96, 1995. Les Dix-huitièmes Journées Arithmétiques (Bordeaux, 1993).

[14] Richard T. Bumby. Sums of four squares. In *Number theory (New York, 1991–1995)*, pages 1–8. Springer, New York, 1996.

[15] Claude Chevalley. Démonstration d'une hypothèse de M. Artin. *Abh. Math. Sem. Hamburg*, 11:73–75, 1936.

[16] Henri Cohen. *A course in computational algebraic number theory*, volume 138 of *Graduate Texts in Mathematics*. Springer-Verlag, Berlin, 1993.

[17] Stephen D. Cohen. Explicit theorems on generator polynomials. *Finite Fields Appl.*, 11(3):337–357, 2005.

[18] Harvey Cohn. *A classical invitation to algebraic numbers and class fields*. Springer-Verlag, New York, 1978. With two appendices by Olga Taussky: "Artin's 1932 Göttingen lectures on class field theory" and "Connections between algebraic number theory and integral matrices", Universitext.

[19] David A. Cox. *Primes of the form $x^2 + ny^2$*. A Wiley-Interscience Publication. John Wiley & Sons Inc., New York, 1989. Fermat, class field theory and complex multiplication.

[20] V. B. Dem′yanov. On representation of a zero of forms of the form $\sum_{i=1}^{m} a_i x_i^n$. *Dokl. Akad. Nauk SSSR (N.S.)*, 105:203–205, 1955.

[21] É. Galois. Sur la théorie des nombres. *Bulletin des sciences mathématiques de Ferussac*, XIII, §218, Juin 1830. Pp. 113–127 in: Écrits et mémoires mathéma-tiques d'Évariste Galois, Robert Bourgne et J.-P. Azra (eds), Gauthier-Villars & Cie, Paris, 1962.

[22] Joachim von zur Gathen and Jürgen Gerhard. *Modern computer algebra*. Cambridge University Press, Cambridge, second edition, 2003.

[23] Joachim von zur Gathen, Igor Shparlinski, and Alistair Sinclair. Finding points on curves over finite fields. *SIAM J. Comput.*, 32(6):1436–1448 (electronic), 2003.

[24] Carl Friedrich Gauss. *Disquisitiones arithmeticae*. Gerh. Fleischer Iun., Leipzig, 1801. English translation by Arthur A. Clarke, Springer-Verlag, New York, 1986.

[25] Marvin J. Greenberg. *Lectures on forms in many variables*. W. A. Benjamin, Inc., New York-Amsterdam, 1969.

[26] Larry C. Grove. *Classical groups and geometric algebra*, volume 39 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, RI, 2002.

[27] Tom Hansen and Gary L. Mullen. Primitive polynomials over finite fields. *Math. Comp.*, 59(200):639–643, S47–S50, 1992.

[28] G.H. Hardy and E.M. Wright. *An introduction to the theory of numbers*. Oxford, at the Clarendon Press, 1965. Fourth edition, 3rd corrected printing.

[29] Mark van Hoeij. Factoring polynomials and the knapsack problem. *J. Number Theory*, 95(2):167–189, 2002.

[30] Jean-René Joly. Équations et variétés algébriques sur un corps fini. *Enseignement Math. (2)*, 19:1–117, 1973.

[31] Neal Koblitz. *A course in number theory and cryptography*, volume 114 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, second edition, 1994.

[32] T. Y. Lam. *The algebraic theory of quadratic forms*. W. A. Benjamin, Inc., Reading, Mass., 1973. Mathematics Lecture Note Series.

[33] Serge Lang. *Algebra*, volume 211 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, third edition, 2002.

[34] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261(4):515–534, 1982.

[35] H. W. Lenstra, Jr. Finding isomorphisms between finite fields. *Math. Comp.*, 56(193):329–347, 1991.

[36] Rudolf Lidl and Harald Niederreiter. *Finite fields*, volume 20 of *Encyclopedia of Mathematics and its Applications*. Addison-Wesley Publishing Company Advanced Book Program, Reading, MA, 1983. With a foreword by P. M. Cohn.

[37] J.H. van Lint. *Introduction to coding theory*, volume 86 of *Graduate Texts in Mathematics*. Springer-Verlag, Berlin, third edition, 1999.

[38] Karl K. Norton. *Numbers with small prime factors, and the least kth power non-residue.* Memoirs of the American Mathematical Society, No. 106. American Mathematical Society, Providence, R.I., 1971.

[39] Richard S. Pierce. *Associative algebras*, volume 88 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1982. Studies in the History of Modern Science, 9.

[40] A. Rényi. Representations for real numbers and their ergodic properties. *Acta Math. Acad. Sci. Hungar.*, 8:477–493, 1957.

[41] Lajos Rónyai. Computing the structure of finite algebras. *J. Symbolic Comput.*, 9(3):355–373, 1990.

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

[43] René Schoof. Counting points on elliptic curves over finite fields. *J. Théor. Nombres Bordeaux*, 7(1):219–254, 1995. Les Dix-huitièmes Journées Arithmétiques (Bordeaux, 1993).

[44] Štefan Schwarz. On universal forms in finite fields. *Časopis Pěst. Mat. Fys.*, 75:45–50, 1950.

[45] Andrew Shallue and Christiaan E. van de Woestijne. Construction of rational points on elliptic curves over finite fields. In *Algoritmic Number Theory (ANTS VII)*, Lecture Notes in Comput. Sci., Berlin, 2006. Springer-Verlag. To appear.

[46] Daniel Shanks. Five number-theoretic algorithms. In *Proceedings of the Second Manitoba Conference on Numerical Mathematics (Univ. Manitoba, Winnipeg, Man., 1972)*, pages 51–70. Congressus Numerantium, No. VII, Winnipeg, Man., 1973. Utilitas Math.

[47] Joseph H. Silverman. *The arithmetic of elliptic curves*, volume 106 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1992. Corrected reprint of the 1986 original.

[48] Denis Simon. Quadratic equations in dimensions 4, 5 and more. Preprint. URL: `http://www.math.unicaen.fr/~simon/maths/dim4.html`, 2005.

[49] Denis Simon. Solving quadratic equations using reduced unimodular quadratic forms. *Math. Comp.*, 74(251):1531–1543 (electronic), 2005.

[50] M. Skałba. Points on elliptic curves over finite fields. *Acta Arith.*, 117(3):293–301, 2005.

[51] Bart de Smit. Primitive elements in integral bases. *Acta Arith.*, 71(2):159–170, 1995.

[52] Alberto Tonelli. Bemerkung über die Auflösung quadratischer Congruenzen. *Nachr. Göttingen*, (10):344–346, 1891. Reported in Dickson's History, Vol. 1, Ch. VII, item 193, p. 215.

[53] Leonard Tornheim. Sums of n-th powers in fields of prime characteristic. *Duke Math. J.*, 4:359–362, 1938.

[54] Daqing Wan. Generators and irreducible polynomials over finite fields. *Math. Comp.*, 66(219):1195–1212, 1997.

[55] E. Warning. Bemerkung zur vorstehenden Arbeit von Herrn Chevalley. *Abh. Math. Sem. Hamburg*, 11:76–83, 1936.

[56] André Weil. Numbers of solutions of equations in finite fields. *Bull. Amer. Math. Soc.*, 55:497–508, 1949.

[57] André Weil. *Basic number theory.* Classics in Mathematics. Springer-Verlag, Berlin, 1995. Reprint of the second (1973) edition.

[58] Eric W. Weisstein. Waring's problem. From MathWorld–A Wolfram Web Resource. URL: `http://mathworld.wolfram.com/WaringsProblem.html`.

[59] Arne Winterhof. On Waring's problem in finite fields. *Acta Arith.*, 87(2):171–177, 1998.

[60] Trevor D. Wooley. Diophantine problems in many variables: the role of additive number theory. In *Topics in number theory (University Park, PA, 1997)*, volume 467 of *Math. Appl.*, pages 49–83. Kluwer Acad. Publ., Dordrecht, 1999.

Samenvatting

Het deterministisch oplossen van vergelijkingen over eindige lichamen

De onderstaande twee algoritmische representatieproblemen, die onderling sterk verwant zijn, worden in dit proefschrift opgelost.

Probleem S1 Gegeven een eindig lichaam $\mathbb{F}$, gegeven een positief geheel getal n en gegeven elementen $a_0, a_1, \ldots, a_n$ van $\mathbb{F}$ die alle ongelijk nul zijn, bereken elementen $x_0, x_1, \ldots, x_n$ van $\mathbb{F}$, niet alle nul, zodanig dat

$$\sum_{i=0}^{n} a_i x_i^n = 0.$$

Probleem S2 Gegeven een eindig lichaam $\mathbb{F}$, gegeven een positief geheel getal n, en gegeven elementen $b, a_1, \ldots, a_n$ van $\mathbb{F}$, alle ongelijk nul, bereken elementen $x_1, \ldots, x_n$ van $\mathbb{F}$ zodanig dat

$$\sum_{i=1}^{n} a_i x_i^n = b,$$

of toon aan dat zulke elementen niet bestaan.

Het gaat bij deze problemen respectievelijk om het representeren van nul, danwel van elementen ongelijk nul, door diagonale vormen in vele variabelen over eindige lichamen. De klassieke stelling van Chevalley-Warning vertelt ons dat de vergelijking in Probleem S1 altijd een oplossing heeft — Hoofdstuk 2 geeft hierover meer details. In Probleem S2 kan oplosbaarheid van de vergelijking gegarandeerd worden door te eisen dat *alle elementen van $\mathbb{F}$ te schrijven zijn als sommen van $n$de machten van elementen van $\mathbb{F}$*. In Hoofdstuk 2 wordt, voor zover ik weet voor de eerste maal, aangetoond dat deze voorwaarde voldoende is.

De meeste bewijzen die gegeven worden voor oplosbaarheidsresultaten zoals de zojuist genoemde, leiden niet tot efficiënte methoden om ook daadwerkelijk oplossingen te vinden. Mijn hoofdresultaat is de constructie van twee efficiënte algoritmen, voor beide problemen één, die zulke oplossingen inderdaad berekenen. Het bewijs van dit resultaat beslaat het gehele proefschrift en bevat een gedetailleerde beschrijving en analyse van deze twee sterk verwante algoritmen.

Stelling S3 *Er zijn deterministische algoritmen voor het oplossen van Problemen S1 en S2 waarvan de looptijd polynomiaal is in termen van n en $\log q$, waarbij q gelijk is aan het aantal elementen van $\mathbb{F}$.*

Tot nu toe hebben diverse efficiënte algoritmen voor het oplossen van niet-lineaire vergelijkingen over eindige lichamen essentieel gebruik gemaakt van probabilistische elementen. Deze gerandomiseerde deelstappen dienen meestal om een element van de multiplicatieve groep van het beschouwde lichaam te vinden dat buiten een gegeven ondergroep valt, zoals een element dat geen kwadraat is. Het vernieuwende van het hier gepresenteerde resultaat is dat deze gerandomiseerde technieken geëlimineerd zijn: de methoden van dit proefschrift zijn zuiver deterministisch.

De algoritmen van dit proefschrift gebruiken alleen elementaire technieken, maar hun formulering kent vele stappen en maakt gebruik van subroutines op diverse niveaus. Als hulpresultaten die op zichzelf van belang zijn, geef ik methoden om, voor een gegeven n, een voortbrenger voor een eindig lichaam over zijn priemlichaam te geven die een $n$de macht is, en om een gegeven element van een eindig lichaam te schrijven als som van $n$de machten met ten hoogste n termen — of, in beide gevallen, om vast te stellen dat deze taken onmogelijk zijn (resp. Hoofdstukken 4 en 5). Een andere hoeksteen is een deterministische aanpassing van het worteltrekalgoritme van Tonelli-Shanks, waarover meer te vinden is in Hoofdstuk 3.

Uit de beschrijving van mijn algoritmen zal duidelijk worden dat de door mij gegeven versies alreeds praktisch bruikbaar zijn. De analyse van hun rekentijd, die niet moeilijk is, laat zien dat hun prestaties nauwelijks minder zijn dan die van probabilistische oplossingsmethoden voor Problemen S1 en S2.

Ik geef toepassingen op verschillende algoritmische problemen op het gebied van eindige lichamen, waaronder het berekenen van een rationaal punt op een kwadratisch hyperoppervlak, het berekenen van isomorfismen tussen kwadratische ruimten, het berekenen van lichaamsvoortbrengers met voorgeschreven norm, het trivialiseren van centrale simpele algebra's van graad 2. Wellicht onverwacht, tenslotte, is het feit dat mijn algoritmen de 'missing link' blijken te zijn in de constructie van een efficient deterministisch algoritme voor het berekenen van rationale punten op elliptische krommen over eindige lichamen.

Een implementatie van alle in dit proefschrift ontwikkelde algoritmen in de computeralgebrataal Magma is bij de auteur verkrijgbaar.

Dankwoord

Om te beginnen dank ik de directie van het Mathematisch Instituut van de Universiteit Leiden voor mijn aanstelling als Assistent in Opleiding, waardoor ik van een van mijn favoriete activiteiten, het oplossen van wiskundige problemen, mijn werk kon maken. Ook dank ik hen voor de mogelijkheid deze aanstelling in deeltijd te vervullen, waardoor er ruimte bleef voor een andere hobby, de muziek.

Ik heb op het Instituut een inspirerende en leerzame tijd gehad, die begonnen is met de studies wiskunde en informatica, en die nu na vele jaren tot een einde komt. Het was een plezier op te trekken met de vele collega's, in het bijzonder in de groepen Meetkunde en topologie en Algebra en getaltheorie, die elkaar dagelijks troffen aan de lunchtafel, maar ook daarbuiten. Op de tweewekelijkse Intercity-seminars zagen we ook de collega's van andere Nederlandse (en ook enkele Belgische) universiteiten; deze intensieve uitwisseling is me altijd uitstekend bevallen.

Dank dus aan mijn oud-afstudeerdocent Rob Tijdeman, bij wie ik altijd voor advies terecht kon; aan Nils, voor de kennismaking met computeralgebrapakketten die ook echte algebra kunnen; aan Richard en Willem-Jan, voor hun deskundigheid op Linuxgebied waarvan ik telkens weer kon profiteren; aan Reinier voor vele interessante discussies en een cruciale verwijzing; aan Frans, voor zijn betrokkenheid en de muzikale uitwisselingen; aan alle andere collega's in Leiden; en daarbuiten, aan Mascha Honsbeek, Chris Zaal, Jaap Top, Gert-Jan van der Heiden, Ronald van Luijk, Jasper Scholten en Benne de Weger.

I've met people too numerous to mention in many international conferences and other meetings around the world, but especially in Berkeley and Lausanne, each of whom contributed to the pleasure of being a mathematician. I hope I will meet several of you again!

Das letzte Halbjahr, das ich am Radon-Institut der ÖAW in Linz verbracht habe, habe ich sehr genossen.

Ein besonderes Dankeschön gilt Arne Winterhof, für die Einladung nach Linz, und für deine Freundschaft. Ich wünsche dir jetzt alles Gute im Laufbahn, und hoffentlich sehen wir uns noch oft.

Ich danke nebendem Josef Schicho und Bruno Buchberger, die mir die Teilnahme am Sondersemester über Gröbnerbasen ermöglicht haben.

Ich danke Tanja Löhr (Mainz) für die Gestaltung des Umschlags dieser Doktorarbeit, und hoffe, Oliver und dich bald wieder zu sehen.

Een van de effecten van het lidmaatschap van een studentenvereniging en van een kerkelijke gemeente is dat het je vele vrienden oplevert die geen wiskundigen zijn. Het voordeel en het nadeel daarvan is dat je gedwongen wordt je niet alleen met je favoriete activiteiten en hobby's, maar ook met het echte leven bezig te houden. Ga zo door, zou ik zeggen, en bedankt voor alle goede momenten die we kunnen delen! Ik hoop dat we, ook nu ik in Oostenrijk ben gaan wonen, gelegenheid zullen vinden om elkaar te zien.

Dank in het bijzonder aan Gertjan en Annemieke, voor jullie vriendschap en gastvrijheid, het uitzoeken van stapels post tijdens mijn afwezigheid, en aan Gertjan voor het bekleden van de functie van paranimf.

Tenslotte is daar mijn familie: mijn ouders, zus, broers, schoonzus, en de verdere familieleden. Veel dank voor jullie morele en materiële steun gedurende mijn hele leven, en niet minder in de afgelopen tijd, waarin ik wel wat steun kon gebruiken; ik kan altijd op jullie rekenen! We kunnen elkaar tegenwoordig allemaal per e-mail bereiken, maar gelukkig komen we elkaar ook nog wel eens in het echt tegen, zelfs in Oostenrijk. Mijn broer Pieter was ook paranimf; bedankt daarvoor!

Ik dank God voor het feit dat ik een bevoorrecht leven kan leiden, zeker in vergelijking met vele anderen op de wereld. Ik geniet er elke dag van in Zijn wereld rond te lopen en zoveel mooie dingen tegen te komen, of het nu vogels, cartoons, klanken, mensen, wiskundige stellingen of bergen zijn. Ik dank Hem voor zijn leiding en vraag Hem om een plaats in de wereld waar ik voor mijn medemensen zinvol kan zijn.

Curriculum vitae

Christiaan van de Woestijne werd op 18 september 1975 geboren te Rotterdam. Hij bezocht van 1986 tot 1992 de Reformatorische Scholengemeenschap “Guido de Brès” te Rotterdam, waar hij het gymnasiumdiploma behaalde.

In de zomer van 1992 werd hij winnaar van de Nationale Biologie-Olympiade. Vervolgens nam hij deel aan de 3^e^ Internationale Biologie-Olympiade in Poprad, Slowakije, waar hij een bronzen medaille behaalde.

In het jaar 1992-1993 studeerde hij aan de Evangelische Hogeschool te Amersfoort (het Basisjaar-programma), waar onder meer de filosofiecolleges van prof. dr. dr. dr. W. J. Ouweneel zijn denken gevormd hebben.

Vanaf 1993 studeerde hij wiskunde en informatica aan de Universiteit Leiden. In 1998 behaalde hij hier met lof het doctoraaldiploma wiskunde met de scriptie “On the diameter of tuples of weighted powers”. Begeleiders waren dr. B. M. M. de Weger en prof. dr. R. Tijdeman.

In 1999 behaalde hij het doctoraaldiploma informatica, eveneens met lof, met de scriptie “A formal characterisation of the Delilah system”, onder begeleiding van dr. H. J. Hoogeboom en dr. C. L. J. M. Cremers.

Van 1999 tot 2003 was hij aangesteld als Assistent in Opleiding aan het Mathematisch Instituut van de Universiteit Leiden, waar hij promotie-onderzoek verrichtte onder begeleiding van prof. dr. H. W. Lenstra, Jr., en tevens vele werkcolleges en enkele hoorcolleges gaf. Financiële bijdragen uit de Spinozaprijs die in 1999 aan prof. Lenstra werd toegekend, en van de Europese Unie, stelden hem in staat enkele maanden te verblijven aan het MSRI te Berkeley, California, Verenigde Staten, en aan de EPFL te Lausanne, Zwitserland.

Tevens was hij in deze tijd werkzaam als deeltijd-docent aan de lerarenopleiding wiskunde 1^e^ graads aan de Christelijke Hogeschool De Driestar te Gouda en aan de Hogeschool Rotterdam.

Van 1998 tot 2005 fungeerde Christiaan als organist van de Scots International Church te Rotterdam en vervangend organist van de Marekerk te Leiden. Om zijn vaardigheden te verbeteren, studeerde hij van 2000 tot 2005 orgel aan het Rotterdams Conservatorium bij Aart Bergwerff, welke studie hij op 14 juni 2005 afsloot met het behalen van het diploma Klassieke muziek Eerste fase.

In het najaar van 2004 stelde het Mathematisch Instituut in Leiden hem voor vier maanden aan als onderzoeksassistent. In deze periode werd zijn promotie-onderzoek, waaruit dit proefschrift is voortgekomen, afgerond, de redactie van het proefschrift echter nog niet.

Van oktober 2005 tot mei 2006 werkte hij als wiskundig onderzoeker aan het Radon Instituut van de Oostenrijkse Academie van Wetenschappen te Linz en het Research Institute for Symbolic Computation (RISC) van de Johannes Kepler Universität Linz. Vanaf 1 juni 2006 is hij werkzaam als Postdoc aan de Technische Universität Graz, Oostenrijk.