Western Graduate&PostdoctoralStudies

Western University
Scholarship@Western

Electronic Thesis and Dissertation Repository

9-23-2015 12:00 AM

Computing in Algebraic Closures of Finite Fields

Javad Doliskani
The University of Western Ontario

Supervisor
Eric Schost
The University of Western Ontario Joint Supervisor
Jan Minac
The University of Western Ontario

Graduate Program in Computer Science

A thesis submitted in partial fulfillment of the requirements for the degree in Doctor of Philosophy

Recommended Citation

Doliskani, Javad, "Computing in Algebraic Closures of Finite Fields" (2015). *Electronic Thesis and Dissertation Repository*. 3282.
https://ir.lib.uwo.ca/etd/3282

Computing in Algebraic Closures of Finite Fields

Javad Doliskani

Supervisor
Eric Schost
The University of Western Ontario

Computing in Algebraic Closures of Finite Fields

(Thesis format: Integrated-Article)

Javad Doliskani

Department of Computer Science

A thesis submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

The School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada

Abstract

We present algorithms to construct and perform computations in algebraic closures of finite fields. Inspired by algorithms for constructing irreducible polynomials, our approach for constructing closures consists of two phases; First, extension towers of prime power degree are built, and then they are glued together using composita techniques. To be able to move elements around in the closure we give efficient algorithms for computing isomorphisms and embeddings. In most cases, our algorithms which are based on polynomial arithmetic, rather than linear algebra, have quasi-linear complexity.

Keywords: Algebraic closure, polynomial arithmetic, finite field.

Co-Authorship

This document is based on the following joint works with Éric Schost and Luca Defeo.

Chapter 2 Taking roots over high extensions of finite fields. *Mathematics of Computation*, 83(285), pp. 435-446, 2014.

Chapter 3 Computing in degree 2k-extensions of finite fields of odd characteristic. *Des. Codes Cryptography*, 74(3), pp. 559-569, 2015.

Chapter 4 Fast algorithms for ℓ-adic towers over finite fields. *In Proceedings of the 38th International Symposium on Symbolic and Algebraic Computation (ISSAC'13).* ACM, New York, NY, USA.

Chapter 5 Fast arithmetic for the algebraic closure of finite fields. *In Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation (ISSAC '14)*, ACM, New York, NY, USA.

List of Tables

List of Figures

Contents

Chapter 1

Introduction

The theory of finite fields has found much attention during recent decades, most notably due to developments in the branches of computer science such as cryptography, coding theory, switching circuits, and combinatorics [18]. In applications, one almost always encounters finite fields that are extensions of the so-called prime fields. For example, in public-key cryptography, there are cryptosystems that are based on the group of points on an algebraic curve. To have a secure system one must go through the process of choosing a random secure curve. A good class of candidates for such curves are hyperelliptic curves. One of the efficient methods of choosing a secure hyperelliptic curve requires counting the number of points on the curve, which in turn requires building a large number of successive extensions over a small prime field [10, 11].
The natural need for extensions usually arises from the need for manipulating polynomials and their roots. Once the extension are built, the next step is to be able to move elements from one extension to the other. This is where algebraic closures come to the fore. From a computational point of view, an algebraic closure of a given field can be seen as a large enough finite extension that contains the roots of a given finite set of polynomials. Computation in algebraic closures has been considered by others, e.g. [24] and references therein. In this chapter, we briefly review some basic properties of finite fields, and discuss some notations on the running time complexities of some basic arithmetics over them. At the end, we give an overview of our computational approach to algebraic closures.

1.1 Notations

In this section, we give a very brief review of finite fields concepts and notations used in the subsequent chapters. For a more detailed preliminary on finites fields we refer the reader to Appendix A. Given a prime number p, we denote the prime finite field with p elements by $\mathbb{F}_p$. The prime number p is called the characteristic of the field.

Irreducible polynomials. The univariate polynomial ring over $\mathbb{F}_p$ is denoted by $\mathbb{F}_p[X]$. The elements of $\mathbb{F}_p[x]$ are of the form $f(X) = a_n X^n + a^{n-1} X^{n-1} + \cdots + a_0$ where $a_i \in \mathbb{F}_p$. The degree of f, denoted by $\deg f$, is n. We call f monic if its leading coefficient a_n is 1. The division and remainder operation in $\mathbb{F}_p[X]$ is done using the usual polynomial arithmetic. A polynomial $f \in \mathbb{F}_p[X]$ is reducible if it can be written as $f = gh$ for polynomials $g, h \in \mathbb{F}_p[X]$ of degree > 0;

otherwise it is called *irreducible*. For a monic irreducible polynomials f of degree n the quotient $\mathbb{F}_p[X]/\langle f\rangle$ is a finite field extension of $\mathbb{F}_p$ of degree n. We usually denote such an extension by $\mathbb{F}_q$ where $q = p^n$.

Operation complexities The nature of the algorithms presented in this document suggests an algebraic complexity model. This means we will count operations $\{+,-,\times,\div\}$ in a based field such as $\mathbb{F}_p$ or $\mathbb{F}_q$. Consequently, we have to choose a representation for elements of field extensions. For example, one could proceed implementing algorithms using normal basis representation, in which case the complexity estimates are stated differently. However, we shall use the monomial basis representation, since in particular our implementations and result comparisons are based on systems such as NTL [23], Magma [3], and FLINT [12]. In the monomial representation, the finite field or even ring extension are represented by quotients. For example, the finite field $\mathbb{F}_{q^n}$ is represented by $\mathbb{F}_q[X]/\langle f\rangle$ where f is a monic irreducible polynomial of degree n in $\mathbb{F}_q[X]$. So an element $a \in \mathbb{F}_{q^n}$ is a polynomial of degree less than n in $\mathbb{F}_q[X]$.
One of the most fundamental operations in all algebraic systems is *polynomial multiplication*. We denote the upper bound for the cost of multiplying two polynomials of degree n by $\mathsf{M}(n)$. This means for a given field K and two polynomials $f, g \in K[X]$ of degree n, fg can be computed using $\mathsf{M}(n)$ multiplications in K. The best known upper bound for $\mathsf{M}(n)$ is $O(n\log(n)\log\log(n))$ [19, 6]. This is done using Fast Fourier Transform. The idea is to reduce the polynomial multiplication to point-wise vector-vector multiplication using an invertible change of representation. See [27, Chapter 8] for more details. We assume that $\mathsf{M}(n)$ is a super-linear function, i.e. not bounded by any linear function.
Another fundamental operation is *matrix multiplication*. Given two square matrices of size n over a field K we assume they can be multiplied using $O(n^\omega)$ multiplications in K, where ω is called the linear algebra exponent. A naive implementation would give $\omega = 3$. A slightly better algorithm due to Strassen [25] gives a bound $\omega \leq 2.81$. The algorithm uses a divide and conquer approach to reduce the multiplication problem to sub-blocks of the given matrices. The smallest known bound is $\omega \leq 2.37$ due to Coppersmith and Winograd [7].
Given polynomials $f, g, h \in K[X]$ of degree n, the *modular composition* problem is to compute $f(g) \mod h$. We denote the upper bound for modular composition by $\mathsf{C}(n)$. In a boolean complexity model, in which bit or word operation are counted, there is an algorithm due to Kedlaya and Umans [15, 26] that runs in $O(n^{1+\varepsilon}\log(p)^{1+o(1)})$ operations. Since we do not follow a boolean model, and there is still no competitive implementation of their algorithm that we know of, we opt for the well-known algorithm of Brent and Kung [5]. Their algorithm gives the bound $O(n^{(\omega+1)/2})$. We can take $\omega \leq 2.37$ using the Coppersmith and Winograd algorithm which gives $\mathsf{C}(n) = O(n^{1.69})$. There is the slightly better bound $\mathsf{C}(n) = O(n^{1.67})$ due to Pan [13], which uses rectangular matrix multiplication.

Algebraic closures. A field extension $K \subseteq L$ is said to be algebraic if every element $a \in L$ is a root of a monic polynomial in $K[X]$. A field K is algebraically closed if $L = K$ for every algebraic extension of L. An algebraic closure of a field K, denoted by $\overline{K}$, is an algebraically closed algebraic extension of K.
Let E, F be extensions of K. Assume that E, F are contained in some larger field L. We define the *compositum* of E, F, denoted by EF, to be the smallest subfield of L containing both E, F.

Generally, we can define the compositum of a family $\{F_i\}_{i\in I}$ of subfields of L to be the smallest subfield of L containing all F_i. The compositum of E, F is not defined unless we have embeddings of both fields into a common field L. Let $m = [E : K]$ and $n = [F : K]$. We have the following diagram of extensions:

$$\begin{tikzcd}
& L \arrow[d, dash] & \\
& EF \arrow[dl, dash, "\leq n"'] \arrow[dr, dash, "\leq m"] & \\
E \arrow[dr, dash, "m"'] & & F \arrow[dl, dash, "n"] \\
& K &
\end{tikzcd}$$

From the diagram $[EF : K] \leq mn$ and $m \mid [EF : K]$ and $n \mid [EF : K]$. So if $\gcd(m,n) = 1$ then $[EF : K] = mn$. Therefore, if E, F are finite fields $\mathbb{F}_{p^n}, \mathbb{F}_{p^m}$ with $\gcd(m,n) = 1$ then the compositum EF is the finite field $\mathbb{F}_{p^{mn}}$. This allows us to define the algebraic closure of $\mathbb{F}_p$ as the union

$$\overline{\mathbb{F}_p} = \bigcup_{i\geq 1} \mathbb{F}_{p^i}.$$

1.2 Our approach

Let $\mathbb{F}_q$ be a finite field with $q = p^n$ a prime power. We divide the construction of $\overline{\mathbb{F}_q}$ into two major steps: building towers, and gluing towers. Let us explain what these mean. Let ℓ be a prime number, and let

$$\mathbb{F}_q \subset \mathbb{F}_{q^\ell} \subset \mathbb{F}_{q^{\ell^2}} \subset \cdots$$

a sequence of extensions each of degree ℓ. We call this an *ℓ-adic tower*. Define the *ℓ-adic closure* of $\mathbb{F}_q$ to be the union

$$\mathbb{F}_q^{(\ell)} = \bigcup_{i\geq 0} \mathbb{F}_{q^{\ell^i}}.$$

Assume we have build these towers for different prime ℓ. Then we can glue them together to get the algebraic closure. By gluing here we mean computing composita. Figure 1.1 shows the above structure of $\overline{\mathbb{F}_q}$. The solid circles in the middle are the composita.

ℓ-adic towers. Let $F_i = \mathbb{F}_{q^{\ell^i}}$ be the level i of the ℓ-adic tower for an integer $i > 0$. The goal is to be able to compute in F_i in quasi-linear time in the extension degree ℓ^i. For this we need first to construct the tower, and then be able to move up and down to different levels of the tower efficiently.

In many cases, building extensions and isomorphisms of finite fields requires taking roots of elements. In Chapter 2 we discuss taking arbitrary roots over extensions of finite fields. The idea is

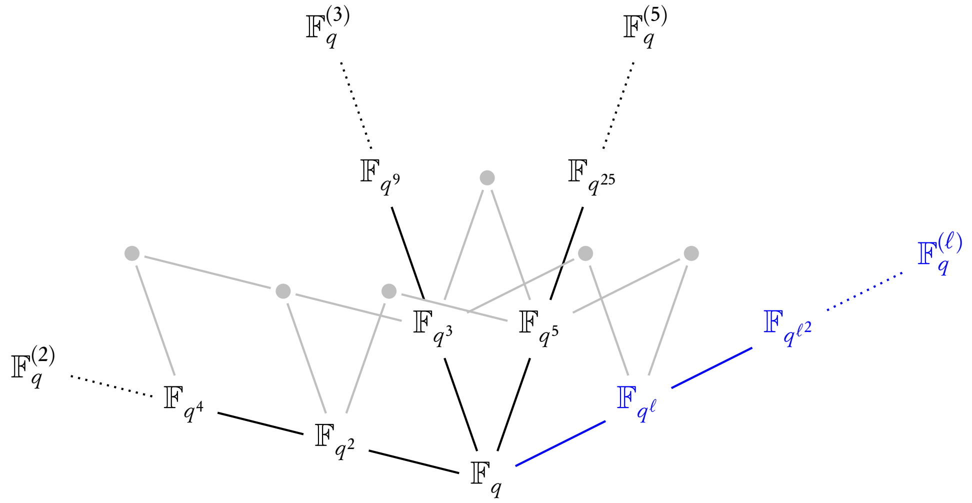


Figure 1.1: Algebraic closure of $\mathbb{F}_q$

to reduce the problem to root extraction in subfields using trace-like computations. For example, using our algorithm, the complexity of taking square roots in $\mathbb{F}_q$ is an expected $O(\mathsf{M}(n)\log(p)+\mathsf{C}(n)\log(n))$ operations in $\mathbb{F}_p$. Computing roots in finite fields was previously considered by others like Cipolla [28] and Tonelli-Shanks [20]. Kaltofen and Shoup [14] give an Equal Degree Factorization (EDF) algorithm that can be used for computing roots.

The case $\ell = 2$ of the ℓ-adic towers, which is of particular interest, is treated separately in Chapter 3. Our algorithms presented there for construction of the tower and performing basic operations inside the tower are all quasi-linear time. The following table summarizes the complexities of our algorithms for some well-known operations for this case.

Operation	Cost
addition / subtraction	$O(n)$
multiplication	$\mathsf{M}(n)+O(n)$
inversion	$O(\mathsf{M}(n))$
quadratic residuosity	$O(\mathsf{M}(n)+\log(q))$
square root	$O(\mathsf{M}(n)\log(nq))$ (expected)
isomorphism	$O(n+\log(n)\log(q))$ (expected)

Note the improvement of the square root complexity here compared to the one for general finite fields which has a $\mathsf{C}(n)$ term. Another special case of the ℓ-adic towers is $\ell = p$, called the Artin-Schreier tower. The algorithms for this case were presented in [9].

Chapter 4 discusses the towers for a general ℓ. The field $\mathbb{F}_{q^{\ell^i}}$ can be represented by two natural bases over $\mathbb{F}_q$. For a univariate representation of the form $\mathbb{F}_q[X_i]/\langle Q_i\rangle$, where $Q_i \in \mathbb{F}_q[X_i]$ is a monic irreducible polynomial of degree ℓ^i, we get the monomial basis

$$\mathbf{U}_i = (1, x_i, x_i^2, \ldots, x_i^{\ell^i - 1})$$

for $\mathbb{F}_{q^{\ell^i}}$ over $\mathbb{F}_q$. Here x_i is the residue class of X_i modulo Q_i. But we can also consider $\mathbb{F}_{q^{\ell^i}}$ as a degree ℓ extension over $\mathbb{F}_{q^{\ell^{i-1}}}$, i.e. as a bivariate quotient

$$\mathbb{F}_q[X_{i-1},X_i]/\langle Q_{i-1}(X_{i-1}), T_i(X_{i-1},X_i)\rangle$$

where T_i is a monic polynomial of degree ℓ in X_i, and of degree less than ℓ^{i-1} in X_{i-1}. This gives the basis

$$\mathbf{B}_i = (1,\ldots,x_{i-1}^{\ell^{i-1}-1},\ldots,x_i^{\ell-1},\ldots,x_{i-1}^{\ell^{i-1}-1}x_i^{\ell-1}).$$

for $\mathbb{F}_{q^{\ell^i}}$ over $\mathbb{F}_q$. The complexity of constructing the tower, which requires some initialization and finding Q_i, T_i, is close to quasi-linear. Using the usual algorithms, basic operations like multiplication and inversion in $\mathbb{F}_q[X_i]/\langle Q_i\rangle$ are done using respectively $O(\mathsf{M}(\ell^i))$ and $O(\mathsf{M}(\ell^i)\log(\ell^i))$ operations in $\mathbb{F}_q$. These are quasi-linear in the extension degree ℓ^i. Moving up and down in the tower is done using repeated applications of two operations called *lift* and *push*. Lifting refers to the change of basis from $\mathbf{B}_i$ to $\mathbf{U}_i$ while pushing is the inverse transformation. These operations are also very close to being quasi-linear. The following table summarizes our main complexity results.

Condition	Initialization	Q_i, T_i	Lift, push
$q = 1 \bmod \ell$	$O_e(\log(q))$	$O(\ell^i)$	$O(\ell^i)$
$q = -1 \bmod \ell$	$O_e(\log(q))$	$O(\ell^i)$	$O(\mathsf{M}(\ell^i)\log(\ell^i))$
$-$	$O_e(\ell^2+\mathsf{M}(\ell)\log(q))$	$O(\mathsf{M}(\ell^{i+1})\mathsf{M}(\ell)\log(\ell^i)^2)$	$O(\mathsf{M}(\ell^{i+1})\mathsf{M}(\ell)\log(\ell^i))$
$4\ell \leq q^{1/4}$	$\tilde{O_e}(\ell\log^5(q)+\ell^3)$ (bit)	$O_e(\ell^2+\mathsf{M}(\ell)\log(\ell q)+\mathsf{M}(\ell^i)\log(\ell^i))$	$O(\mathsf{M}(\ell^i)\log(\ell^i))$
$4\ell \leq q^{1/4}$	$\tilde{O_e}(\ell\log^5(q))$ (bit) $+ O_e(\mathsf{M}(\ell)\sqrt{q}\log(q))$	$O_e(\log(q)+\mathsf{M}(\ell^i)\log(\ell^i))$	$O(\mathsf{M}(\ell^i)\log(\ell^i))$

The probabilistic complexities with expected running time are denoted by $O_e(\ )$. Also $\tilde{O_e}(\ )$ indicates the additional omission of logarithmic factors. Although in some cases there are extra factors of ℓ, we have achieved quasi-linear time in the degree of the extension ℓ^i in most of the cases. Our algorithm for constructing towers are inspired by the previous works of Shoup [21, 22] and Lenstra / De Smit [16], and Couveignes / Lercier [8]. See the introduction of Chapter 4 for more details.

Composita. In Chapter 5, we discuss the composita of fields. Let $\mathbb{F}_{p^m} = \mathbb{F}_p[x]/\langle Q_m(x)\rangle$ and $\mathbb{F}_{p^n} = \mathbb{F}_p[y]/\langle Q_n(y)\rangle$ be two finite fields where $Q_m(x)\in\mathbb{F}_p[x]$ and $Q_n(y)\in\mathbb{F}_p[y]$ are irreducible of comprime degrees $m, n > 1$ respectively. Define the *composed product* of Q_m, Q_n as

$$Q_{mn}(z) = \prod_{\substack{1\leq i\leq m\\ 1\leq j\leq n}} (z - a_i b_j)$$

where $(a_i)_{1\leq i\leq m}$ and $(b_j)_{1\leq j\leq n}$ are roots of Q_m and Q_n in the algebraic closure of $\mathbb{F}_p$. The polynomial Q_{mn} is irreducible of degree mn in $\mathbb{F}_p[z]$, see [4]. The field $\mathbb{F}_p[z]/\langle Q_{mn}(z)\rangle$ is the compositum of $\mathbb{F}_{p^m}$ and $\mathbb{F}_{p^n}$, and there exists embeddings φ_x, φ_y, and an isomorphism Φ of the form

$$\begin{array}{rrcl}
\varphi_x: & \mathbb{F}_p[x]/\langle Q_m\rangle & \to & \mathbb{F}_p[z]/\langle Q_{mn}\rangle,\\
\varphi_y: & \mathbb{F}_p[y]/\langle Q_n\rangle & \to & \mathbb{F}_p[z]/\langle Q_{mn}\rangle,\\
\Phi: & A = \mathbb{F}_p[x,y]/\langle Q_m, Q_n\rangle & \to & \mathbb{F}_p[z]/\langle Q_{mn}\rangle\\
 & xy & \longleftarrow & z.
\end{array}$$

The goal is to compute φ_x, φ_y, and Φ and Φ^{-1} efficiently. We present algorithms for computing φ_x and φ_y which run using $O(n\mathsf{M}(m)+m\mathsf{M}(n))$ operations in $\mathbb{F}_p$. Assuming that $m \leq n$, Φ and Φ^{-1} can be computed using either $O(m^2\mathsf{M}(n))$ or $O(\mathsf{M}(mn)n^{1/2}+\mathsf{M}(m)n^{(\omega+1)/2})$ operations in $\mathbb{F}_p$ where ω is the linear algebra exponent. Therefore, computing embeddings φ_x, φ_y is quasi-linear in mn while computing the isomorphisms Φ, Φ^{-1} has an extra factor of m or n. Previous algorithms such as the ones presented in [2], and [17, 1] also compute embeddings of finite fields. They rely on linear algebra which results in complexities at least quadratic in the degree of the extensions.

Bibliography

[1] B. Allombert. Explicit computation of isomorphisms between finite fields. *Finite Fields Appl.*, 8(3):332 – 342, 2002.

[2] W. Bosma, J. Cannon, and A. Steel. Lattices of compatibly embedded finite fields. *J. Symb. Comput.*, 24(3-4):351–369, 1997.

[3] Wieb Bosma, John Cannon, and Catherine Playoust. The MAGMA algebra system I: the user language. *J. Symbolic Comput.*, 24(3-4):235–265, 1997.

[4] J.V. Brawley and L. Carlitz. Irreducibles and the composed product for polynomials over a finite field. *Discrete Mathematics*, 65(2):115 – 139, 1987.

[5] R. P. Brent and H. T. Kung. Fast algorithms for manipulating formal power series. *Journal of the Association for Computing Machinery*, 25(4):581–595, 1978.

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

[7] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comp*, 9(3):251–280, 1990.

[8] Jean-Marc Couveignes and Reynald Lercier. Fast construction of irreducible polynomials over finite fields. *To appear in the Israel Journal of Mathematics*, July 2011.

[9] L. De Feo and É. Schost. Fast arithmetics in Artin-Schreier towers over finite fields. *Journal of Symbolic Computation*, 47(7):771–792, 2012.

[10] Luca De Feo. Fast algorithms for computing isogenies between ordinary elliptic curves in small characteristic. *Journal of Number Theory*, 131(5):873–893, May 2011.

[11] P. Gaudry and É. Schost. Point-counting in genus 2 over prime fields. *J. Symbolic Comput.*, 47(4):368–400, 2012.

[12] William Hart. Fast library for number theory: an introduction. *Mathematical Software–ICMS 2010*, pages 88–91, 2010.

[13] X. Huang and V. Y. Pan. Fast rectangular matrix multiplication and applications. *J. Complexity*, 14(2):257–299, 1998.

[14] E. Kaltofen and V. Shoup. Fast polynomial factorization over high algebraic extensions of finite fields. In *ISSAC'97*, pages 184–188. ACM, 1997.

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

[16] Hendrick W. Lenstra and Bart De Smit. Standard models for finite fields: the definition, 2008.

[17] H. W. Lenstra Jr. Finding isomorphisms between finite fields. *Math. Comp.*, 56(193):329–347, 1991.

[18] Rudolf Lidl and Günter Pilz. *Applied abstract algebra*. Springer Science & Business Media, 2012.

[19] A. Schönhage and V. Strassen. Schnelle Multiplikation grosser Zahlen. *Computing*, 7:281–292, 1971.

[20] D. Shanks. Five number-theoretic algorithms. In *Proceedings of the Second Manitoba Conference on Numerical Mathematics*, pages 51–70, 1972.

[21] Victor Shoup. New algorithms for finding irreducible polynomials over finite fields. *Math. Comp.*, 54:435–447, 1990.

[22] Victor Shoup. Fast construction of irreducible polynomials over finite fields. *J. Symbolic Comput.*, 17(5):371–391, 1994.

[23] Victor Shoup. NTL: A library for doing number theory. `http://www.shoup.net/ntl`, 2003.

[24] Allan K. Steel. Computing with algebraically closed fields. *Journal of Symbolic Computation*, 45(3):342 – 372, 2010.

[25] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969.

[26] C. Umans. Fast polynomial factorization and modular composition in small characteristic. In *STOC*, pages 481–490, 2008.

[27] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 2003.

[28] H. C. Williams. Some algorithms for solving $x^q \equiv N \pmod{p}$. In *Proceedings of the Third Southeastern Conference on Combinatorics, Graph Theory and Computing (Florida Atlantic Univ., Boca Raton, Fla., 1972)*, pages 451–462. Florida Atlantic Univ., 1972.

Chapter 2

Taking Roots over High Extensions of Finite Fields

2.1 Introduction

Beside its intrinsic interest, computing m-th roots over finite fields (for m an integer at least equal to 2) has found many applications in computer science. Our own interest comes from elliptic and hyperelliptic curve cryptography; there, square root computations show up in pairing-based cryptography [3] or point-counting problems [8].

Our result in this paper is a new algorithm for computing m-th roots in a degree n extension $\mathbb{F}_q$ of the prime field $\mathbb{F}_p$, with p a prime. Our emphasis is on the case where p is thought to be small, and the degree n grows. Roughly speaking, we reduce the problem to m-th root extraction in a lower degree extension of $\mathbb{F}_p$ (when $m = 2$, we actually reduce the problem to square root extraction over $\mathbb{F}_p$ itself).

Our complexity model. It is possible to describe the algorithm in an abstract manner, independently of the choice of a basis of $\mathbb{F}_q$ over $\mathbb{F}_p$. However, to give concrete complexity estimates, we have to decide which representation we use, the most usual choices being monomial and normal bases. We choose to use a monomial basis, since in particular our implementation is based on the library NTL [22], which uses this representation. Thus, the finite field $\mathbb{F}_q = \mathbb{F}_{p^n}$ is represented as $\mathbb{F}_p[X]/\langle f \rangle$, for some monic irreducible polynomial $f \in \mathbb{F}_p[X]$ of degree n; elements of $\mathbb{F}_q$ are represented as polynomials in $\mathbb{F}_p[X]$ of degree less than n. We will briefly mention the normal basis representation later on.

The costs of all algorithms are measured in number of operations $+, \times, \div$ in the base field $\mathbb{F}_p$ (that is, we are using an algebraic complexity model) — at the end of this introduction, we discuss how our results can be stated in the boolean model, in light especially of results by Umans [25] and Kedlaya and Umans [13].

We shall denote upper bounds for the cost of *polynomial multiplication* and *modular composition* by respectively $\mathsf{M}(n)$ and $\mathsf{C}(n)$. This means that over any field $\mathbb{K}$, we can multiply polynomials of degree n in $\mathbb{K}[X]$ in $\mathsf{M}(n)$ base field operations, and that we can compute $f(g) \bmod h$ in $\mathsf{C}(n)$ operations in $\mathbb{K}$, when f, g, h are degree n polynomials. We additionally require that both M and

C are super-linear functions, as in [26, Chapter 8], and that $\mathsf{M}(n) = O(\mathsf{C}(n))$. In particular, since we work in the monomial basis, multiplications and inversions in $\mathbb{F}_q$ can be done in respectively $O(\mathsf{M}(n))$ and $O(\mathsf{M}(n)\log(n))$ operations in $\mathbb{F}_p$, see again [26].

The best known bound for $\mathsf{M}(n)$ is $O(n\log(n)\log\log(n))$, achieved by using Fast Fourier Transform [19, 5]. The most well-known bound for $\mathsf{C}(n)$ is $O(n^{(\omega+1)/2})$, due to Brent and Kung [4], where ω is such that matrices of size n over any field $\mathbb{K}$ can be multiplied in $O(n^{\omega})$ operations in $\mathbb{K}$; this estimate assumes that $\omega > 2$, otherwise some logarithmic terms may appear. Using the algorithm of Coppersmith and Winograd [6], we can take $\omega \le 2.37$ and thus $\mathsf{C}(n) = O(n^{1.69})$; an algorithm by Huang and Pan [10] actually achieves a slightly better exponent of 1.67, by means of rectangular matrix multiplication.

Main result. We will focus in this paper on the case of t-th root extraction, where t is a prime divisor of $q-1$; the general case of m-th root extraction, with m arbitrary, can easily be reduced to this case (see the discussion after Theorem 20).

The core of our algorithm is a reduction of t-th root extraction in $\mathbb{F}_q$ to t-th root extraction in an extension of $\mathbb{F}_p$ of smaller degree. Our algorithm is probabilistic of Las Vegas type, so its running time is given as an expected number of operations. With this convention, our main result is the following.

Theorem 1. *Let t be a prime factor of $q-1$, with $q = p^n$, and let s be the order of p in $\mathbb{Z}/t\mathbb{Z}$. Given $a \in \mathbb{F}_q^*$, one can decide if a is a t-th power in $\mathbb{F}_q^*$, and if so compute one of its t-th roots, by means of the following operations:*

- *an expected $O(s\mathsf{M}(n)\log(p) + \mathsf{C}(n)\log(n))$ operations in $\mathbb{F}_p$;*
- *a t-th root extraction in $\mathbb{F}_{p^s}$.*

Thus, we replace t-th root extraction in a degree n extension by a t-th root extraction in an extension of degree $s \le \min(n,t)$. The extension degree s is the largest one for which t still divides $p^s - 1$, so iterating the process does not bring any improvement: the t-th root extraction in $\mathbb{F}_{p^s}$ must be dealt with by another algorithm. The smaller s is, the better.

A useful special case is $t = 2$, that is, we are taking square roots; the assumption that t divides $q-1$ is then satisfied for all odd primes p and all n. In this case, we have $s = 1$, so the second step amounts to square root extraction in $\mathbb{F}_p$. Since this can be done in $O(\log(p))$ expected operations in $\mathbb{F}_p$, the total running time of the algorithm is an expected $O(\mathsf{M}(n)\log(p) + \mathsf{C}(n)\log(n))$ operations in $\mathbb{F}_p$.

A previous algorithm by Kaltofen and Shoup [12] allows one to compute t-th roots in $\mathbb{F}_{p^n}$ in expected time $O((\mathsf{M}(t)\mathsf{M}(n)\log(p) + t\mathsf{C}(n) + \mathsf{C}(t)\mathsf{M}(n))\log(n))$; we discuss it further in the next section. This algorithm requires no assumption on t, so it can be used in our algorithm in the case $s > 1$, for t-th root extraction in $\mathbb{F}_{p^s}$. Then, its expected running time is $O((\mathsf{M}(t)\mathsf{M}(s)\log(p) + t\mathsf{C}(s) + \mathsf{C}(t)\mathsf{M}(s))\log(s))$.

The strategy of using Theorem 20 to reduce from $\mathbb{F}_q$ to $\mathbb{F}_{p^s}$ then using the Kaltofen-Shoup algorithm over $\mathbb{F}_{p^s}$ is never more expensive than using the Kaltofen-Shoup algorithm directly over $\mathbb{F}_q$. For $t = O(1)$, both strategies are within a constant factor; but even for the smallest case $t = 2$, our algorithm has advantages (as explained in the last section). For larger t, the gap in our favor will increase for cases when s is small (such as when t divides $p-1$, corresponding to $s=1$).

Finally, let us go back to the remark above, that for any m, one can reduce m-th root extraction of $a \in \mathbb{F}_q^*$ to computing t-th roots, with t dividing $q-1$; this is well known, see for instance [1, Chapter 7.3]. We write $m = uv$ with $(v, q-1) = 1$ and $t \mid q-1$ for every prime divisor t of u, and we assume that a is indeed an m-th power.

- We first compute the v-th root a_0 of a as $a_0 = a^{v^{-1} \bmod q-1}$ by computing the inverse ℓ of $v \bmod q-1$, and computing an ℓ-th power in $\mathbb{F}_q$. This takes $O(n\mathsf{M}(n)\log(p))$ operations in $\mathbb{F}_p$.
- Let $u = \prod_{i=1}^d m_i^{\alpha_i}$ be the prime factorization of u, which we assume is given to us. Then, for $k = 1, \ldots, \alpha_1$, we compute an m_1-th root a_k of a_{k-1} using Theorem 20, so that a_{α_1} is an $m_1^{\alpha_1}$-th root of a_0.

  One should be careful in the choice of the m_1-th roots (which are not unique), so as to ensure that each a_k is indeed an u/m_1^i-th power: if the given a_k is not such a power, we can multiply it by a m_1-th root of unity until we find a suitable one. The root of unity can be found by the algorithm of Theorem 20.

  Once we know a_{α_1}, the same process can be applied to compute an $m_2^{\alpha_2}$-th root of a_{α_1}, and so on.

The first step, taking a root of order v, may actually be the bottleneck of this scheme. When v is small compared to n, it may be better to use here as well the algorithm by Kaltofen and Shoup mentioned above.

In a boolean model. In our algebraic model, when n grows, the bottleneck of our algorithm (or of the Kaltofen-Shoup algorithm) is modular composition, since there is currently no known algorithm with cost quasi-linear in n.

If we analyze running times in a boolean model (counting bit or word operations), much better can be done: Kedlaya and Umans [13], following previous work by Umans [25], give an algorithm with a boolean cost that grows like $n^{1+\varepsilon}\log(p)^{1+o(1)}$ for modular composition in degree n over $\mathbb{F}_p$, for any given $\varepsilon > 0$. They also show that minimal polynomial of elements of $\mathbb{F}_q = \mathbb{F}_{p^n}$ can be computed for the same cost.

The algorithms of the following sections can be analyzed in the boolean model without difficulty (for definiteness, over a boolean RAM with logarithmic access cost — the Kedlaya-Umans algorithm uses table lookup in large tables). The only differences are in the cost of modular composition over $\mathbb{F}_p$, as well as minimal polynomial computation in $\mathbb{F}_q$, for which we use the results by Kedlaya and Umans. Using the fact that arithmetic in $\mathbb{F}_p$ can be done in boolean time $\log(p)^{1+o(1)}$, the running time reported in Theorem 20 then becomes $O(s\mathsf{M}(n)\log(p)^{2+o(1)} + n^{1+\varepsilon}\log(p)^{1+o(1)})$ bit operations, for any $\varepsilon > 0$; this admits the upper bound $O(sn^{1+\varepsilon}\log(p)^{2+o(1)})$. With respect to the extension degree n, this is close to being linear time.

From the practical point of view, however, we did not use the Kedlaya-Umans algorithm in our experiments, since we do not have a competitive implementation of it (and currently know of no such implementation).

Organization of the chapter. The next section reviews and discusses known algorithms; Section 2.3 gives the details of the root extraction algorithm and some experimental results. In all the paper, $(\mathbb{F}_q^*)^t$ denotes the set of t-th powers in $\mathbb{F}_q^*$.

Acknowledgments. We thank NSERC and the Canada Research Chairs program for financial support. We also thank the referee for very helpful remarks.

2.2 Previous work

Let t be a prime factor of $q-1$. In the rest of this section, we discuss previous algorithms for t-th root extraction, with a special focus on the case $t = 2$ (square roots), which has attracted most attention in the literature. Note that our assumptions exclude the case of pth root extraction in characteristic p.

We shall see in Section 2.3 that given a prime t as above, the cost of testing for t-th power is always dominated by the t-th root extraction; thus, for an input $a \in \mathbb{F}_q^*$, we always assume that $a \in (\mathbb{F}_q^*)^t$.

All algorithms discussed below rely on some form of exponentiation in $\mathbb{F}_q$, or in an extension of $\mathbb{F}_q$, with exponents that grow linearly with q. As a result, a direct implementation using binary powering uses $O(\log(q))$ multiplications in $\mathbb{F}_q$, that is, $O(n\mathsf{M}(n)\log(p))$ operations in $\mathbb{F}_p$. Even using fast multiplication, this is quadratic in n; alternative techniques should be used to perform the exponentiation, when possible.

Some special cases of square root computation. If G is a group with an odd order s, then the mapping $f : G \to G$, $f(a) = a^2$ is an automorphism of G; hence, every element $a \in G$ has a unique square root, which is $a^{(s+1)/2}$. Thus, if $q \equiv 3 \pmod 4$, the square root of any $a \in (\mathbb{F}_q^*)^2$ is $a^{(q+1)/4}$; this is because $(\mathbb{F}_q^*)^2$ is a group of odd order $(q-1)/2$.

More complex schemes allow one to compute square roots for some increasingly restricted classes of prime powers q. The following table summarizes results known to us; in each case, the algorithm uses $O(1)$ exponentiations and $O(1)$ additions / multiplications in $\mathbb{F}_q$. The table indicates what exponents are used in the exponentiations.

Table 2.1: Some special cases for square roots

Algorithm	q	exponent
folklore	3 (mod 4)	$(q+1)/4$
Atkin	5 (mod 8)	$(q-5)/8$
Müller [15]	9 (mod 16)	$(q-1)/4$ and $(q-9)/16$
Kong *et al.* [14]	9 (mod 16)	$(q-9)/8$ and $(q-9)/16$

As was said above, using a direct binary powering approach to exponentiation, all these algorithms use $O(n\mathsf{M}(n)\log(p))$ operations in $\mathbb{F}_p$. Some extensions to higher roots have appeared in the literature, for example to cube roots in the case where $q = 7 \pmod 9$ [17], with similar running times.

Cipolla's square root algorithm. To compute the square root of $a \in (\mathbb{F}_q^*)^2$, Cipolla's algorithm uses an element b in $\mathbb{F}_q$ such that $b^2 - 4a$ is not a square in $\mathbb{F}_q$. Then, the polynomial $f(Y) = Y^2 - bY + a$ is irreducible over $\mathbb{F}_q$, hence $\mathbb{K} = \mathbb{F}_q[Y]/\langle f \rangle$ is a field. Let y be the residue class of Y modulo $\langle f \rangle$. Since f is the minimal polynomial of y over $\mathbb{F}_q$, $N_{\mathbb{K}/\mathbb{F}_q}(y) = a$, ensuring that $\sqrt{a} = Y^{(q+1)/2} \bmod (Y^2 - bY + a)$.
Finding a quadratic nonresidue of the form $b^2 - 4a$ by choosing a random $b \in \mathbb{F}_q$ requires an expected $O(1)$ attempts [1, page 158]. The quadratic residue test, and the norm computation take $O(\mathsf{M}(n)\log(n) + \log(p))$ and $O(n\mathsf{M}(n)\log(p))$ multiplications in $\mathbb{F}_p$ respectively. Therefore, the cost of the algorithm is an expected $O(n\mathsf{M}(n)\log(p))$ operations in $\mathbb{F}_p$.
Algorithms extending Cipolla's to the computation of t-th roots in $\mathbb{F}_p$, where t is a prime factor of $p-1$, are in [27, 28, 16].

The Tonelli-Shanks algorithm. We will describe the algorithm in the case of square roots, although the ideas extend to higher orders. Tonelli's algorithm [23] and Shanks' improvement [20] use discrete logarithms to reduce the problem to a subgroup of $\mathbb{F}_q^*$ of odd order. Let $q - 1 = 2^r \ell$ with $(\ell, 2) = 1$ and let H be the unique subgroup of $\mathbb{F}_q^*$ of order ℓ. Assume we find a quadratic nonresidue $g \in \mathbb{F}_q^*$; then, the square root of $a \in \mathbb{F}_q^*$ can be computed as follows: we can express a as $g^s h \in g^s H$ by solving a discrete logarithm in $\mathbb{F}_q^*/H$; s is necessarily even, so that $\sqrt{a} = g^{s/2} h^{(\ell+1)/2}$.
According to [18], the discrete logarithm requires $O(r^2 \mathsf{M}(n))$ multiplications in $\mathbb{F}_p$; all other steps take $O(n\mathsf{M}(n)\log(p))$ operations in $\mathbb{F}_p$. Hence, the expected running time of the algorithm is $O((r^2 + n\log(p))\mathsf{M}(n))$ operations in $\mathbb{F}_p$. Thus, the efficiency of this algorithm depends on the structure of $\mathbb{F}_q^*$: there exists an infinite sequence of primes for which the cost is $O(n^2 \mathsf{M}(n)\log(p)^2)$, see [24]. An extension to the computation of cube roots modulo p is in [17], with a similar cost analysis.

Improving the exponentiation. All algorithms seen before use at best $O(n\mathsf{M}(n)\log(p))$ operations in $\mathbb{F}_p$, because of the exponentiation. Using ideas going back to [11], Barreto *et al.* [3] observed that for some of the cases seen above, the exponentiation can be improved, giving a cost subquadratic in n.
For instance, when taking square roots with $q = 3 \pmod 4$, the exponentiation $a^{(q+1)/4}$ can be reduced to computing $a^{1+u+\cdots+u^{(n-3)/2}}$, with $u = p^2$, plus two (cheap) exponentiations with exponents $p(p-1)$ and $(p+1)/4$. The special form of the exponent $1 + u + \cdots + u^{(n-3)/2}$ makes it possible to apply a binary powering approach, involving $O(\log(n))$ multiplications and exponentiations, with exponents that are powers of p.
Further examples for square roots are discussed in [14, 9], covering the entries of Table 2.1. These references assume a normal basis representation; using (as we do) the monomial basis and modular composition techniques (which will be explained in the next section), the costs become $O(\mathsf{M}(n)\log(p) + \mathsf{C}(n)\log(n))$. Some cases of higher index roots are in [2]: if t is a factor of

$p-1$, such that t^2 does not divide $p-1$, and if $\gcd(n,t)=1$, then t-th root extraction can be done using $O(t\mathsf{M}(n)\log(p)+\mathsf{C}(n)\log(n))$ operations in $\mathbb{F}_p$.

Kaltofen and Shoup's algorithm. Finally, we mention what is, as far as we know, the only algorithm achieving an expected subquadratic running time in n (using the monomial basis representation), without any assumption on p.
Consider a factor t of $q-1$. To compute a t-th root of $a \in (\mathbb{F}_q^*)^t$, the idea is simply to factor $Y^t - a \in \mathbb{F}_q[Y]$ using polynomial factorization techniques. Since we know that a is a t-th power, this polynomial splits into linear factors, so we can use an Equal Degree Factorization (EDF) algorithm.
A specialized EDF algorithm, dedicated to the case of high-degree extension of a given base field, was proposed by Kaltofen and Shoup [12]. It mainly reduces to the computation of a trace-like quantity $b + b^p + \cdots + b^{p^{n-1}}$, where b is a random element in $\mathbb{F}_q[Y]/\langle Y^t - a\rangle$. Using a binary powering technique similar to the one of the previous paragraph, this results in an expected running time of $O((\mathsf{M}(t)\mathsf{M}(n)\log(p) + t\mathsf{C}(n) + \mathsf{C}(t)\mathsf{M}(n))\log(n))$ operations in $\mathbb{F}_p$; remark that this estimate is faster than what is stated in [12] by a factor $\log(t)$, since here we only need one root, instead of the whole factorization. In the particular case $t = 2$, this becomes $O((\mathsf{M}(n)\log(p)+\mathsf{C}(n))\log(n))$. This achieves a running time subquadratic in n.
This idea actually allows one to compute a t-th root, for arbitrary t: starting from the polynomial $Y^t - a$, we apply the above algorithm to $\gcd(Y^t - a, Y^q - Y)$; computing Y^q modulo $Y^t - a$ can be done by the same binary powering techniques.

2.3 A new root extraction algorithm

In this section, we focus on t-th root extraction in $\mathbb{F}_q$, for t a prime dividing $q-1$ (as we saw in Section 3.1, m-th root extraction, for an integer $m \geq 2$, reduces to taking t-th roots, where t is a prime factor of m dividing $q-1$).
The algorithm we present uses the trace $\mathbb{F}_q \to \mathbb{F}_{q'}$, for some subfield $\mathbb{F}_{q'} \subset \mathbb{F}_q$ to reduce t-th root extraction in $\mathbb{F}_q$ to t-th root extraction in $\mathbb{F}_{q'}$. We assume as before that the field $\mathbb{F}_q$ is represented by a quotient $\mathbb{F}_p[X]/\langle f\rangle$, with $f(X) \in \mathbb{F}_p[X]$ a monic irreducible polynomial of degree n. We let x be the residue class of X modulo $\langle f\rangle$.
Since we will handle both $\mathbb{F}_q$ and $\mathbb{F}_{q'}$, conversions may be needed. We recall that the minimal polynomial $g \in \mathbb{F}_p[Z]$ of an element $b \in \mathbb{F}_q$ can be computed in $O(\mathsf{C}(n)+\mathsf{M}(n)\log(n))$ operations in $\mathbb{F}_p$ [21]. Then, $\mathbb{F}_{q'} = \mathbb{F}_p[Z]/\langle g\rangle$ is a subfield of $\mathbb{F}_q = \mathbb{F}_p[X]/\langle f\rangle$; given $r \in \mathbb{F}_{q'}$, written as a polynomial in Z, we obtain its representation on the monomial basis of $\mathbb{F}_q$ by means of a modular composition, in time $\mathsf{C}(n)$. We will write this operation $\mathrm{Embed}(r, \mathbb{F}_q)$. Note that when b is in $\mathbb{F}_p$, all these operations are actually free.

2.3.1 An auxiliary algorithm

We first discuss a binary powering algorithm to solve the following problem. Starting from $\lambda \in \mathbb{F}_q$, we are going to compute

$$\alpha_i(\lambda) = \lambda^{1+p^s} + \lambda^{1+p^s+p^{2s}} + \cdots + \lambda^{1+p^s+p^{2s}+\cdots+p^{is}}$$

for given integers $i, s > 0$. This question is similar to (but distinct from) some exponentiations and trace-like computations we discussed before; our solution will be a similar binary powering approach, which will perform $O(\log(i))$ multiplications and exponentiations by powers of p. Let

$$\xi_i = x^{p^{is}}, \quad \zeta_i(\lambda) = \lambda^{p^s+p^{2s}+\cdots+p^{is}} \quad \text{and} \quad \delta_i(\lambda) = \lambda^{p^s} + \lambda^{p^s+p^{2s}} + \cdots + \lambda^{p^s+p^{2s}+\cdots+p^{is}},$$

where all quantities are computed in $\mathbb{F}_q$, that is, modulo f; for simplicity, in this paragraph, we will write α_i, ζ_i and δ_i. Note that $\alpha_i = \lambda\delta_i$, and that we have the following relations:

$$\xi_1 = x^{p^s}, \quad \zeta_1 = \lambda^{p^s}, \quad \delta_1 = \lambda^{p^s}$$

and

$$\xi_i = \begin{cases} \xi_{i/2}^{p^{is/2}} & \text{if } i \text{ is even} \\ \xi_{i-1}^{p^s} & \text{if } i \text{ is odd,} \end{cases} \quad \zeta_i = \begin{cases} \zeta_{i/2}\zeta_{i/2}^{p^{is/2}} & \text{if } i \text{ is even} \\ \zeta_1\zeta_{i-1}^{p^s} & \text{if } i \text{ is odd,} \end{cases} \quad \delta_i = \begin{cases} \delta_{i/2} + \zeta_{i/2}\delta_{i/2}^{p^{is/2}} & \text{if } i \text{ is even} \\ \delta_{i-1} + \zeta_i & \text{if } i \text{ is odd.} \end{cases}$$

Because we are working in a monomial basis, computing exponentiations to powers of p is not a trivial task; we will perform them using the following modular composition technique from [7]. Take $j \geq 0$ and $r \in \mathbb{F}_q$, and let R and Ξ_j be the canonical preimages of respectively r and ξ_j in $\mathbb{F}_p[X]$; then, we have

$$r^{p^{js}} = R(\Xi_j) \bmod f,$$

see for instance [26, Chapter 14.7]. We will simply write this as $r(\xi_j)$, and note that it can be computed using one modular composition, in time $\mathsf{C}(n)$. These remarks give us the following recursive algorithm, where we assume that $\xi_1 = x^{p^s}$ and $\zeta_1 = \lambda^{p^s}$ are already known.

Algorithm 1 XiZetaDelta($\lambda, i, \xi_1, \zeta_1$)

```
Input: $\lambda$, a positive integer $i$, $\xi_1 = x^{p^s}$, $\zeta_1 = \lambda^{p^s}$
Output: $\xi_i, \zeta_i, \delta_i$
if $i = 1$ then
    return $\xi_1, \zeta_1, \zeta_1$
end if
$j \leftarrow \lfloor i/2 \rfloor$
$\xi_j, \zeta_j, \delta_j \leftarrow$ XiZetaDelta($\lambda, j, \xi_1, \zeta_1$)
$\xi_{2j} \leftarrow \xi_j(\xi_j)$
$\zeta_{2j} \leftarrow \zeta_j \cdot \zeta_j(\xi_j)$
$\delta_{2j} \leftarrow \delta_j + \zeta_j\delta_j(\xi_j)$
if $i$ is even then
```

```
    **return** $\xi_{2j}, \zeta_{2j}, \delta_{2j}$
**end if**
$\xi_i \leftarrow \xi_{2j}(\xi_1)$
$\zeta_i \leftarrow \zeta_1 \cdot \zeta_{2j}(\xi_1)$
$\delta_i \leftarrow \delta_{2j} + \zeta_i$
**return** $\xi_i, \zeta_i, \delta_i$
```

We deduce the following algorithm for computing $\alpha_i(\lambda)$.

Algorithm 2 Alpha(λ, i)

Input: λ, a positive integer i
Output: α_i

```
$\xi_1 \leftarrow x^{p^s}$
$\zeta_1 \leftarrow \lambda^{p^s}$
$\xi_i, \zeta_i, \delta_i \leftarrow$ XiZetaDelta($\lambda, i, \xi_1, \zeta_1$)
**return** $\lambda\delta_i$
```

Proposition 2. *Algorithm 2 computes $\alpha_i(\lambda)$ using $O(\mathsf{C}(n)\log(is)+\mathsf{M}(n)\log(p))$ operations in $\mathbb{F}_p$.*

Proof. To compute x^{p^s} and λ^{p^s} we first compute x^p and λ^p using $O(\log(p))$ multiplications in $\mathbb{F}_q$, and then do $O(\log(s))$ modular compositions modulo f. The depth of the recursion in Algorithm 1 is $O(\log(i))$; each recursive call involves $O(1)$ additions, multiplications and modular compositions modulo f, for a total time of $O(\mathsf{C}(n))$ per recursive call. □

As said before, the algorithm can also be implemented using a normal basis representation. Then, exponentiations to powers of p become trivial, but multiplication becomes more difficult. We leave these considerations to the reader.

2.3.2 Taking t-th roots

We will now give our root extraction algorithm. As said before, we now let t be a prime factor of $q-1$, and we let s be the order of p in $\mathbb{Z}/t\mathbb{Z}$. Then s divides n, say $n = s\ell$.

We first explain how to test for t-th powers. Testing whether $a \in \mathbb{F}_q^*$ is a t-th power is equivalent to testing whether $a^{(q-1)/t} = 1$. Let $\zeta = a^{(p^s-1)/t}$; then $a^{(q-1)/t} = \zeta^{1+p^s+\cdots+p^{(\ell-1)s}}$. Computing ζ requires $O(s\mathsf{M}(n)\log(p))$, and computing $\zeta^{1+p^s+\cdots+p^{(\ell-1)s}}$ using Algorithm 1 requires $O(\mathsf{C}(n)\log(n)+\mathsf{M}(n)\log(p))$ operations in $\mathbb{F}_p$. Therefore, testing for a t-th power takes $O(\mathsf{C}(n)\log(n)+s\mathsf{M}(n)\log(p))$ operations in $\mathbb{F}_p$.

In the particular case when t divides $p-1$, we can actually do better: we have $a^{(q-1)/t} = \mathrm{res}(f,a)^{(p-1)/t}$, where $\mathrm{res}(\cdot,\cdot)$ is the resultant function. The resultant can be computed using $O(\mathsf{M}(n)\log(n))$ operations in $\mathbb{F}_p$, so the whole test can be done using $O(\mathsf{M}(n)\log(n)+\log(p))$ operations in $\mathbb{F}_p$.

In any case, we can now assume that we are given $a \in (\mathbb{F}_q^*)^t$, with t-th root $\gamma \in \mathbb{F}_q$. Defining $\beta = \mathrm{tr}_{\mathbb{F}_q/\mathbb{F}_{q'}}(\gamma)$, where $\mathrm{tr}_{\mathbb{F}_q/\mathbb{F}_{q'}} : \mathbb{F}_q \to \mathbb{F}_{q'}$ is the trace linear form and $q' = p^s$, we have

$$\begin{aligned}\beta &= \sum_{i=0}^{\ell-1} \gamma^{p^{is}} \\ &= \gamma(1 + \gamma^{p^s-1} + \gamma^{p^{2s}-1} + \cdots + \gamma^{p^{(\ell-1)s}-1}) \\ &= \gamma(1 + a^{(p^s-1)/t} + a^{(p^{2s}-1)/t} + \cdots + a^{(p^{(\ell-1)s}-1)/t}). \end{aligned} \tag{2.1}$$

Let $b = 1 + a^{(p^s-1)/t} + a^{(p^{2s}-1)/t} + \cdots + a^{(p^{(\ell-1)s}-1)/t}$, so that Equation 3.3 gives $\beta = \gamma b$. Taking the t-th power in both sides results in the equation $\beta^t = ab^t$ over $\mathbb{F}_{q'}$. Since we know a, and we can compute b, we can thus determine β by t-th root extraction in $\mathbb{F}_{q'}$. Then, if we assume that $b \neq 0$ (or equivalently that $\beta \neq 0$), we deduce $\gamma = \beta b^{-1}$; to resolve the issue that β may be zero, we will replace a by $a' = ac^t$, for a random element $c \in \mathbb{F}_q^*$.

Computing the t-th root of $a'b^t$ in $\mathbb{F}_{q'}$ is done as follows. We first compute the minimal polynomial $g \in \mathbb{F}_p[Z]$ of $a'b^t$, and let z be the residue class of Z in $\mathbb{F}_p[Z]/\langle g\rangle$. Then, we compute a t-th root r of z in $\mathbb{F}_p[Z]/\langle g\rangle$, and embed r in $\mathbb{F}_q$. The computation of r is done by a black-box t-th root extraction algorithm, denoted by $r \mapsto r^{1/t}$.

It remains to explain how to compute b efficiently. Let $\lambda = a^{(p^s-1)/t}$; then, one verifies that $b = 1 + \lambda + \alpha_{\ell-2}(\lambda)$, so we can use the algorithm of the previous subsection. Putting all together, we obtain the following algorithm

Algorithm 3 t-th root in $\mathbb{F}_q^*$

```
Input: $a \in (\mathbb{F}_q^*)^t$
Output: a $t$-th root of $a$
$s \leftarrow$ the order of $p$ in $\mathbb{Z}/t\mathbb{Z}$
$\ell \leftarrow n/s$
repeat
    choose a random $c \in \mathbb{F}_q$
    $a' \leftarrow ac^t$
    $\lambda \leftarrow a'^{(p^s-1)/t}$
    $b \leftarrow 1 + \lambda + \mathrm{Alpha}(\lambda, \ell-2)$
until $b \neq 0$
$g \leftarrow \mathrm{MinimalPolynomial}(a'b^t)$
$\beta \leftarrow z^{1/t}$ in $\mathbb{F}_p[Z]/\langle g\rangle$
return $\mathrm{Embed}(\beta, \mathbb{F}_q)b^{-1}c^{-1}$
```

The following proposition proves Theorem 20.

Proposition 3. *Algorithm 3 computes a t-th root of a using an expected $O(s\mathsf{M}(n)\log(p) + \mathsf{C}(n)\log(n))$ operations in $\mathbb{F}_p$, plus a t-th root extraction in $\mathbb{F}_{q'}$.*

Proof. Note first that $\beta = 0$ means that $\mathrm{tr}_{\mathbb{F}_q/\mathbb{F}_{q'}}(\gamma c) = 0$. There are q/q' values of c for which this is the case, and hence the probability of having a zero trace is $(q/q')/q = 1/q' \leq 1/2$. So we expect to have to choose $O(1)$ elements in $\mathbb{F}_q$ before exiting the repeat ... until loop. Each pass in the loop uses $O(s\mathsf{M}(n)\log(p))$ operations in $\mathbb{F}_p$ to compute a' and λ, and $O(\mathsf{C}(n)\log(n)+\mathsf{M}(n)\log(p))$ operations in $\mathbb{F}_p$ to compute b.

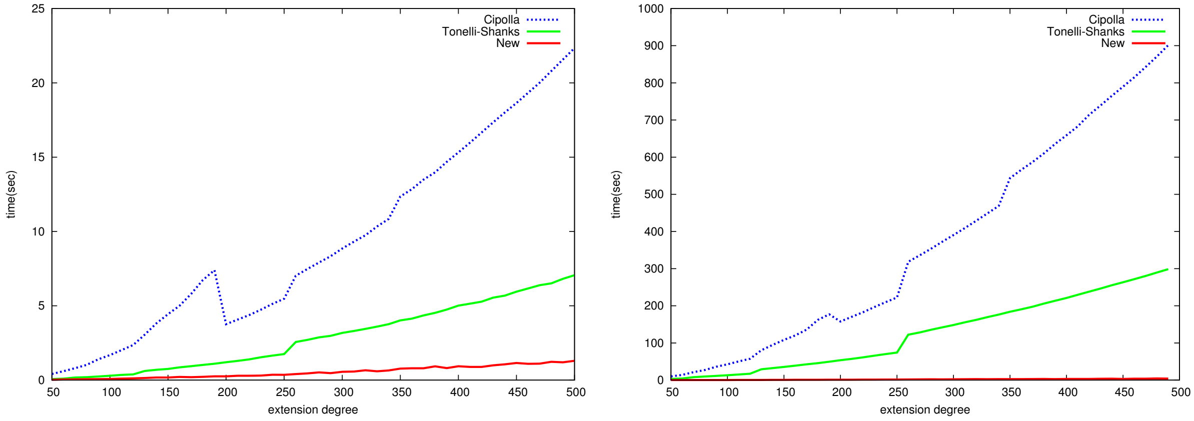
Given a' and b, one obtains b^t and $a'b^t$ using another $O(s\mathsf{M}(n)\log(p))$ operations in $\mathbb{F}_p$; then, computing g takes time $O(\mathsf{C}(n)+\mathsf{M}(n)\log(n))$.

After the black-box call to t-th root extraction modulo g, embedding β in $\mathbb{F}_q$ takes time $\mathsf{C}(n)$. We can then deduce γ by two divisions in $\mathbb{F}_q$, using $O(\mathsf{M}(n)\log(n))$ operations in $\mathbb{F}_p$; this is negligible compared to the cost of all modular compositions. □

2.3.3 Experimental results

We have implemented our root extraction algorithm, in the case $m = 2$ (that is, we are taking square roots); our implementation is based on Shoup's NTL [22]. The experiments in this section were done for the "small" and "large" characteristics $p = 449$ and $p = 348975609381470925634534573457497$, and different values of the extension degree, on a 2.40GHz Intel Xeon. They show that the behavior of our algorithm hardly depends on the characteristic.

Figure 3.1 compares our algorithm to Cipolla's and Tonelli-Shanks' algorithms over $\mathbb{F}_q$, with $q = p^n$, for different values of the extension degree n. For each n, the running time is averaged over five runs, with input being random squares in $\mathbb{F}_q$.



(a) $p = 449$

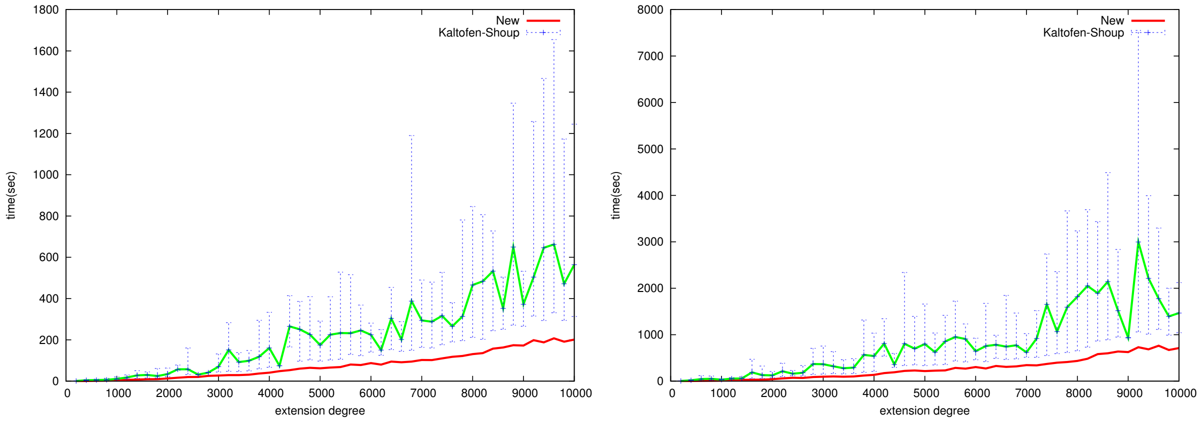
(b) $p = 348975609381470925634534573457497$

Figure 2.1: Our square root algorithm vs. Cipolla's and Tonelli-Shanks' algorithms.

Remember that the bottleneck in Cipolla's and Tonelli-Shanks' algorithms is the exponentiation, which takes $O(n\mathsf{M}(n)\log(p))$ operation in $\mathbb{F}_p$. As it turns out, NTL's implementation of modular composition has $\omega = 3$; this means that with this implementation we have $\mathsf{C}(n) = O(n^2)$, and our algorithm takes expected time $O(\mathsf{M}(n)\log(p)+n^2\log(n))$. Although this implementation is not subquadratic in n, it remains faster than Cipolla's and Tonelli-Shanks' algorithms, in theory and in practice.

Next, Figure 2.2 compares our NTL implementation of the EDF algorithm proposed by Kaltofen and Shoup, and our square root algorithm (note that the range of reachable degrees is much larger that in the first figure). We ran the algorithms for five random elements for each extension degree. At this scale, we observe irregularities in the averaged running time of the Kaltofen-Shoup algorithm, due to its probabilistic behavior.

The vertical dashed lines and the green line respectively show the running time range, and the average running time, of Kaltofen and Shoup's algorithm. In the case of our algorithm (the red graph), the vertical ranges are invisible because the deviation from the average is $\approx 10^{-2}$ seconds.



(a) $p = 449$

(b) $p = 348975609381470925634534573457497$

Figure 2.2: Our algorithm vs. Kaltofen and Shoup's algorithm.

This time, the results are closer. Nevertheless, it appears that the running time of our algorithm behaves more "smoothly", in the sense that random choices seem to have less influence. This is indeed the case. The random choice in Kaltofen and Shoup's algorithm succeeds with probability about $1/2$; in case of failure, we have to run to whole algorithm again. In our case, our choice of an element c in $\mathbb{F}_q^*$ fails with probability $1/p \ll 1/2$; then, there is still randomness involved in the t-th root extraction in $\mathbb{F}_p$, but this step was negligible in the range of parameters where our experiments were conducted.

Another way to express this is to compare the standard deviations in the running times of both algorithms. In the case of Kaltofen-Shoup's algorithm, the standard deviation is about $1/\sqrt{2}$ of the average running time of the whole algorithm. For our algorithm, the standard deviation is no more than $1/\sqrt{p}$ of the average running time of the trace-like computation (which is the dominant part), plus $1/\sqrt{2}$ of the average running time of the root extraction in $\mathbb{F}_p$ (which is cheap).

Finally, we mention that the crossover point between our algorithm and the previous ones varies with p, but is usually small: around $n = 20$ to $n = 30$ for small p (say less than 500) and around $n = 10$ to $n = 20$ for larger values of p. For very small degrees, the Tonelli-Shanks algorithm was the fastest in our experiments. Note that for such small n, big-O analyses lose some significance; this makes it difficult to get accurate theoretical estimates for the behaviour of the various algorithms in this range of degrees.

Bibliography

[1] E. Bach and J. Shallit. *Algorithmic Number Theory; Volume I: Efficient Algorithms*. The MIT Press, 1996.

[2] P. Barreto and J. Voloch. Efficient computation of roots in finite fields. *Designs, Codes and Cryptography*, 39:275–280, 2006.

[3] P. S. L. M. Barreto, H. Y. Kim, B. Lynn, and M. Scott. Efficient algorithms for pairing-based cryptosystems. In *Advances in cryptology—CRYPTO 2002*, volume 2442 of *Lecture Notes in Comput. Sci.*, pages 354–368. Springer, 2002.

[4] R. P. Brent and H. T. Kung. Fast algorithms for manipulating formal power series. *Journal of the Association for Computing Machinery*, 25(4):581–595, 1978.

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

[6] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comp*, 9(3):251–280, 1990.

[7] J. von zur Gathen and V. Shoup. Computing Frobenius maps and factoring polynomials. *Comput. Complexity*, 2(3):187–224, 1992.

[8] P. Gaudry and É. Schost. Genus 2 point counting over prime fields. *Journal of Symbolic Computation*, 47(4):368 – 400, 2012.

[9] D.-G. Han, D. Choi, and H. Kim. Improved computation of square roots in specific finite fields. *IEEE Trans. Comput.*, 58:188–196, 2009.

[10] X. Huang and V. Y. Pan. Fast rectangular matrix multiplication and applications. *J. Complexity*, 14(2):257–299, 1998.

[11] T. Itoh and S. Tsujii. A fast algorithm for computing multiplicative inverses in GF(2^m) using normal bases. *Inform. and Comput.*, 78(3):171–177, 1988.

[12] E. Kaltofen and V. Shoup. Fast polynomial factorization over high algebraic extensions of finite fields. In *ISSAC'97*, pages 184–188. ACM, 1997.

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

[14] F. Kong, Z. Cai, J. Yu, and D. Li. Improved generalized Atkin algorithm for computing square roots in finite fields. *Inform. Process. Lett.*, 98(1):1–5, 2006.

[15] S. Müller. On the computation of square roots in finite fields. *Des. Codes Cryptogr.*, 31(3):301–312, 2004.

[16] N. Nishihara, R. Harasawa, Y. Sueyoshi, and A. Kudo. A remark on the computation of cube roots in finite fields. Cryptology ePrint Archive, Report 2009/457, 2009. `http://eprint.iacr.org/`.

[17] C. Padró and G. Sáez. Taking cube roots in $\mathbb{Z}_m$. *Applied Mathematics Letters*, 15(6):703 – 708, 2002.

[18] S. C. Pohlig and M. E. Hellman. An improved algorithm for computing logarithms over gf(p) and its cryptographic significance. *IEEE Transactions on Information Theory*, IT-24:106–110, 1978.

[19] A. Schönhage and V. Strassen. Schnelle Multiplikation grosser Zahlen. *Computing*, 7:281–292, 1971.

[20] D. Shanks. Five number-theoretic algorithms. In *Proceedings of the Second Manitoba Conference on Numerical Mathematics*, pages 51–70, 1972.

[21] V. Shoup. Fast Construction of Irreducible Polynomials over Finite Fields". *Journal of Symbolic Computation*, 17(5):371–391, May 1994.

[22] V. Shoup. A library for doing number theory (NTL). `http://www.shoup.net/ntl/`, 2009.

[23] A. Tonelli. Bemerkung über die Auflösung quadratischer Congruenzen. *Göttinger Nachrichten*, pages 344–346, 1891.

[24] G. Tornaría. Square roots modulo p. In *LATIN 2002: Theoretical Informatics*, volume 2286 of *Lecture Notes in Comput. Sci.*, pages 430–434. Springer, 2002.

[25] C. Umans. Fast polynomial factorization and modular composition in small characteristic. In *STOC*, pages 481–490, 2008.

[26] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 2003.

[27] H. C. Williams. Some algorithms for solving $x^q \equiv N \pmod{p}$. In *Proceedings of the Third Southeastern Conference on Combinatorics, Graph Theory and Computing (Florida Atlantic Univ., Boca Raton, Fla., 1972)*, pages 451–462. Florida Atlantic Univ., 1972.

[28] K. S. Williams and K. Hardy. A refinement of H. C. Williams' qth root algorithm. *Math. Comp.*, 61(203):475–483, 1993.

Chapter 3

Computing in Degree 2^k-Extensions of Finite Fields of Odd Characteristic

3.1 Introduction

Factoring polynomials and constructing irreducible polynomials are two fundamental operations for finite field arithmetic. As of now, there exists no deterministic polynomial time algorithm for these questions in general, but in some cases better answers can be found. In this chapter, we discuss algorithmic questions arising in such a special case: the construction of, and computations with, extensions of degree $n = 2^k$ of a base field, say $\mathbb{F}_q$, with q a power of an odd prime p. In other words, we are interested with the complexity of computing in the quadratic closure of $\mathbb{F}_q$. There exists a well-known construction of such extensions [11, Th. VI.9.1], which was already put to use algorithmically in [15]: if $q = 1 \bmod 4$, then for any quadratic non-residue $\alpha \in \mathbb{F}_q$, the polynomial $X^{2^k} - \alpha \in \mathbb{F}_q[X]$ is irreducible for any $k \geq 0$, and allows us to construct $\mathbb{F}_{q^{2^k}}$. If $q = 3 \bmod 4$, we first construct a degree-two extension $\mathbb{F}_{q'}$ of $\mathbb{F}_q$, which will thus satisfy $q' = 1 \bmod 4$; this is done by remarking that $X^2 + 1 \in \mathbb{F}_q[X]$ is irreducible, so that we can construct $\mathbb{F}_{q'}$ as $\mathbb{F}_q[X]/\langle X^2 + 1\rangle$.

In this note, taking this remark as a starting point, we give fast algorithms for operations such as multiplication and inversion, trace and norm computation, and most importantly square root computation in $\mathbb{F}_{q^{2^k}}$ (see below for our motivation), as well as isomorphism computation.

We do not make any assumption on the way $\mathbb{F}_q$ is represented: the running time of our algorithms is estimated by counting operations $(+, \times, \div)$ in $\mathbb{F}_q$ at unit cost. The cost of most algorithms will be expressed in terms of the cost of polynomial multiplication. Explicitly, we let $\mathsf{M} : \mathbb{N} \to \mathbb{N}$ be such that degree n polynomials over any ring R can be multiplied in $\mathsf{M}(n)$ operations in R, and such that $\mathsf{M}(n)/n$ is non-decreasing (this will be referred to as *super-linearity*). Using the algorithm of [3], we can take $\mathsf{M}(n) = O(n \log(n) \log\log(n))$.

In view of the discussion above, we will assume that $q = 1 \bmod 4$: if this is not the case, replacing $\mathbb{F}_q$ by $\mathbb{F}_{q'}$ as explained above only induces a constant overhead, since all operations $(+, \times, \div)$ in $\mathbb{F}_{q'}$ can be done using $O(1)$ operations in $\mathbb{F}_q$. Besides, we will assume that the non-quadratic residue α is given; otherwise, such an α can be found by testing an expected $O(1)$ random elements in $\mathbb{F}_q$ for quadratic residuosity. This remains a core non-deterministic component of the construction, since finding α in a polynomial-time deterministic manner is a well-known open question. Some algorithms below are non-deterministic (Las Vegas) as well; for such algorithms, we give the expected running time.

Theorem 4. *Suppose that $q = 1 \bmod 4$. Given a non-quadratic residue $\alpha \in \mathbb{F}_q$, for $k \geq 0$ and $n = 2^k$, the running times for computations in $\mathbb{F}_{q^n}$ reported in Table 3.1 hold.*

Operation	Cost
addition / subtraction	$O(n)$
multiplication	$\mathsf{M}(n) + O(n)$
inversion	$O(\mathsf{M}(n))$
Frobenius	$O(n + \log(q))$
norm	$O(\mathsf{M}(n))$
trace	1
quadratic residuosity	$O(\mathsf{M}(n) + \log(q))$
square root	$O(\mathsf{M}(n)\log(nq))$ (expected)
isomorphism	$O(n + \log(n)\log(q))$ (expected)

Table 3.1: Costs for computations in $\mathbb{F}_{q^n}$, with $n = 2^k$.

This paper can be seen as an analogue of [5], which discusses these questions for Artin-Schreier extensions: the problems we consider, the techniques we use, and the applications (dealing with torsion points of some genus 1 or 2 Jacobians; see below) are similar. More precisely, the recursive techniques used here are similar to those in the reference in terms of exploiting the special structure of the tower and the defining polynomials.

For some questions, such as Frobenius or isomorphism computation, we refer to the next section for a precise description of the operation we perform; however, we mention that in all cases, we use a polynomial basis representation for all computations. In all cases, the combined size of input and output is $O(n)$ elements in $\mathbb{F}_q$, and using fast multiplication, all running times reported here are quasi-linear[1] in n.

Some results in the above table are straightforward (such as addition / subtraction or trace computation) and some are well-known (such as multiplication). Our focus in this note is actually on square root computation, and to the best of our knowledge, this is the first time that such results appear.

The straightforward approaches to square root extraction, using for instance Cipolla's or the Tonelli-Shanks algorithms [16, 4, 13] require a number of multiplications in $\mathbb{F}_{q^n}$ proportional to $\log(q^n)$; the cost is then $O(n\mathsf{M}(n)\log(q))$ operations in $\mathbb{F}_q$, which is at best quadratic in n. It is possible to compute square roots faster, using fast algorithms for *modular composition*, which is the operation that consists in computing $F(G) \bmod H$, given some univariate polynomials F, G, H. Let us denote by $\mathsf{C}(n)$ the cost of this operation, when F, G, H have all degree at most n. Then, the algorithms of [9, 6] compute square roots in $\mathbb{F}_{q^n}$ using $O(\mathsf{M}(n)\log(q) + \mathsf{C}(n)\log(n))$ operations in $\mathbb{F}_q$.

[1] An algorithm is quasi-linear time in n if it has complexity $O(n \log^k n)$ for a constant k.

In our model, where operations in $\mathbb{F}_q$ are counted at unit cost, the best known bound on $\mathsf{C}(n)$ is $\mathsf{C}(n) = O(\sqrt{n}\mathsf{M}(n) + n^{(\omega+1)/2})$, where $2 \le \omega \le 3$ is such that matrices of size m over $\mathbb{F}_q$ can be multiplied in $O(m^\omega)$ operations [2]. Thus, depending on ω, and neglecting logarithmic factors, the cost (with respect to n) of the corresponding square root algorithms ranges between $O(n^{3/2})$ and $O(n^2)$. We remark however that, in a boolean model (on a RAM, using an explicit boolean representation of the elements in $\mathbb{F}_q$, and counting boolean operations at unit cost), Kedlaya and Umans gave in [10] an algorithm of cost $n^{1+\varepsilon}\log(q)^{1+o(1)}$ for modular composition in $\mathbb{F}_q$, for any $\varepsilon > 0$. In that model, algorithms such as those in [9, 6] are thus close to linear time.

None of the algorithms mentioned above is dedicated to the specific case we consider, where the extension degree n is a power of two; few references consider explicitly this particular case [7, 19]. The algorithm of [7] gives a quadratic residuosity test that uses $O(\log(n))$ Frobenius computations and multiplications in $\mathbb{F}_{q^n}$, and $O(\log(q))$ multiplications in $\mathbb{F}_q$. This reference does not specify how to represent the extensions of $\mathbb{F}_q$, and what algorithms should be used for basic operations. Using the algorithms given below for arithmetic and Frobenius computations, the cost of their quadratic residuosity test algorithm is $O((\mathsf{M}(n)+\log(q))\log(n))$ multiplications in $\mathbb{F}_q$; this is slightly slower than our result. The authors of [7] also give algorithms for quadratic residuosity and square root computation for degree 2^k-extensions in their later work [19]. They removed the computation overlap between quadratic residuosity test and square root, but the algorithms still run in times quadratic in n (the quadratic residuosity test being the bottleneck).

This work was inspired by computations with Jacobians of curves of genus 2 over finite fields. Given a genus 2 curve C defined over $\mathbb{F}_p$, the algorithm of [8] computes the cardinality of the Jacobian J of C, following Schoof's elliptic curve point counting algorithm [12]. This involves in particular the computation of successive divisors of 2^k-torsion in J, by means of successive divisions by two in J. Such a division by two boils down to several arithmetic operations, and four square root extractions; thus, the divisors we are computing are defined over the quadratic closure of $\mathbb{F}_p$, or of a small extension of $\mathbb{F}_p$.

In other words, for dealing with 2^k-torsion, the algorithm of [8] relies entirely on the operations above, arithmetic operations and square root extraction. At the time of writing [8], the authors relied on a variant of the Kaltofen-Shoup algorithm [9] with running time $O(\mathsf{M}(n)\log(q) + \mathsf{C}(n)\log(n))$, which was a severe bottleneck; our new algorithms completely alleviate this issue.

After proving Theorem 20 in Section 3.2, we present in Section 3.3 some experiments that confirm that the practical interest of our quasi-linear algorithms for the point-counting problem above.

Acknowledgements. The authors are supported by NSERC and the Canada Research Chairs program. We wish to thank the reviewers for their helpful remarks and suggestions.

3.2 Proof of the complexity statements

In this section, we prove the results stated in Table 3.1. The tower of fields $\mathbb{F}_q, \mathbb{F}_{q^2}, \dots \mathbb{F}_{q^{2^k}}, \dots$ will be written

$$\mathbb{L}_0 \subset \mathbb{L}_1 \subset \cdots \subset \mathbb{L}_k \subset \cdots,$$

with $\mathbb{L}_k = \mathbb{F}_{q^{2^k}}$ for all k.

3.2.1 Representing the fields $\mathbb{L}_k$

The tower of fields $\mathbb{L}_0, \mathbb{L}_1, \ldots$ can be represented in two fashions, using univariate or multivariate representations (see as well [5], for a similar discussion for Artin-Schreier extensions).
Let $\alpha \in \mathbb{F}_q$ be a fixed non-quadratic residue. Define the sequence of polynomials $T_1, T_2, \ldots$ in indeterminates $X_1, X_2, \ldots$ given by

$$T_1 = X_1^2 - \alpha \quad \text{and} \quad T_k = X_k^2 - X_{k-1} \quad \text{for } k > 1,$$

as well as the polynomials $P_1, P_2, \ldots$ given by

$$P_k = X_k^{2^k} - \alpha \quad \text{for } k \geq 1.$$

The discussion in the introduction, as well as the one in [11], shows that for all $k \geq 0$ we have the equality between ideals

$$\langle T_1, \ldots, T_k \rangle = \langle P_k, X_{k-1} - X_k^2, \ldots, X_1 - X_k^{2^{k-1}} \rangle, \tag{3.1}$$

that this ideal (call it A_k) is prime, and that we have

$$\mathbb{L}_k \simeq \mathbb{F}_q[X_1, \ldots, X_k]/A_k.$$

For $k \geq 1$, let x_k be the image of X_k in the residue class ring $\mathbb{F}_q[X_1, \ldots, X_k]/A_k$. Due to the natural embedding of $\mathbb{F}_q[X_1, \ldots, X_k]/A_k$ into $\mathbb{F}_q[X_1, \ldots, X_{k+1}]/A_{k+1}$, x_k can be unequivocally seen as an element of $\mathbb{F}_{q^{2^\ell}}$ for $\ell \geq k$ (and thus as an element of the quadratic closure of $\mathbb{F}_q$).
On the left-hand side of (3.1), we have a Gröbner basis of A_k for the lexicographic order $X_1 < \cdots < X_k$, whereas on the right we have a Gröbner basis for the lexicographic order $X_k < \cdots < X_1$. Corresponding to these two bases of A_k, the elements of $\mathbb{L}_k$ can be represented as polynomials in $x_1, \ldots, x_k$ of degree at most 1 in each variable (and coefficients in $\mathbb{F}_q$), or as polynomials in x_k of degree less than 2^k.
By default, we will use the univariate representation; an element γ of $\mathbb{L}_k$ will thus be written as $\gamma = G(x_k)$, for some polynomial $G \in \mathbb{F}_q[X_k]$ of degree less than 2^k.
We will not explicitly need to convert to multivariate polynomials, but we will often do one step of such a conversion, switching between univariate and bivariate bases. Indeed, for all $k \geq 1$, we have

$$\mathbb{L}_k \simeq \mathbb{F}_q[X_k]/\langle P_k(X_k) \rangle \simeq \mathbb{F}_q[X_{k-1}, X_k]/\langle P_{k-1}(X_{k-1}), X_k^2 - X_{k-1} \rangle.$$

The $\mathbb{F}_q$-monomial basis of $\mathbb{F}_{q^{2^k}}$ associated to the left-hand side is

$$1,\ x_k,\ \ldots,\ x_k^{2^k - 1},$$

whereas the one associated to the right-hand side is

$$1,\ x_{k-1},\ \ldots,\ x_{k-1}^{2^{k-1}-1},\ x_k,\ x_{k-1}x_k,\ \ldots,\ x_{k-1}^{2^{k-1}-1}x_k.$$

The change-of-basis from the univariate basis to the bivariate one amounts to writing an expression $G(x_k)$, with G of degree less than 2^k, as

$$G(x_k) = G_0(x_k^2) + x_k G_1(x_k^2) = G_0(x_{k-1}) + x_k G_1(x_{k-1}),$$

with G_0 and G_1 of degrees less than 2^{k-1}. This does not require any arithmetic operation; the same holds for the converse change-of-basis. Continuing this way on G_0 and G_1, we could convert between univariate and multivariate bases without arithmetic operations if needed.

3.2.2 Arithmetic operations

In this subsection, we discuss the cost of arithmetic operations $(+, \times, \div)$ in $\mathbb{L}_k$, for some $k \geq 0$. In all that follows, we write $n = [\mathbb{L}_k : \mathbb{F}_q]$, that is, $n = 2^k$.

Addition and subtraction take time $O(n)$. For multiplication, using the univariate basis leads to a cost of $\mathsf{M}(n) + O(n)$ operations in $\mathbb{F}_q$, where the $O(n)$ term accounts for reduction modulo P_k (this takes linear time, since P_k is a binomial). Note that a non-trivial (and slightly less efficient) approach using the multivariate representation is in [1].

For inversion, in the univariate basis, a natural idea is to use the fast extended GCD algorithm for univariate polynomials [17, Ch. 11], resulting in a running time $O(\mathsf{M}(n)\log(n))$. However, better can be done by using the tower structure of the fields $\mathbb{L}_k$. Indeed, consider $\gamma = G(x_k) \in \mathbb{L}_k^\times$. Then, writing $G(x_k) = G_0(x_{k-1}) + x_k G_1(x_{k-1})$, we have

$$\begin{aligned}\frac{1}{G(x_k)} &= \frac{1}{G_0(x_{k-1}) + x_k G_1(x_{k-1})} \\ &= \frac{G_0(x_{k-1}) - x_k G_1(x_{k-1})}{G_0(x_{k-1})^2 - x_{k-1} G_1(x_{k-1})^2}.\end{aligned}$$

Therefore, computing an inverse in $\mathbb{L}_k$ amounts to $O(1)$ additions and multiplications in $\mathbb{L}_k$, and one inversion in $\mathbb{L}_{k-1}$. This gives a recursive algorithm with cost $T(k) = T(k-1) + O(\mathsf{M}(2^k))$; the super-linearity of M implies that the running time is $O(\mathsf{M}(2^k)) = O(\mathsf{M}(n))$ operations in $\mathbb{F}_q$.

3.2.3 Frobenius computation

Let γ be in $\mathbb{L}_k$, and let $r = q^d$ for some positive integer d. We explain here how to compute $\gamma^r \in \mathbb{L}_k$; as above, we write $n = 2^k$.

We start with the case $\gamma = x_k$: writing $r = nu + v$ with $0 \leq v < n$, and using the fact that $x_k^n = \alpha$, we obtain $x_k^r = \alpha^u x_k^v$. The constant $\alpha^u = \alpha^{u \bmod (q-1)}$ can be computed in $O(\log(q))$ multiplications in $\mathbb{F}_q$ by repeated squaring. Although our focus is on counting $\mathbb{F}_q$-operations, we also mention how to compute $u \bmod (q-1)$ and v efficiently: first of all, we compute $\rho = r \bmod n(q-1)$ by repeated squaring; then, $u \bmod (q-1)$ and v are respectively the quotient and remainder in the division of ρ by n. They can be computed with a boolean cost polynomial (and actually, quasi-linear) in $\log(d)\log(nq)$, since the bottleneck is an exponentation with exponent $O(d\log(q))$, modulo an integer of bit size $O(\log(nq))$.

For a general γ of the form $\gamma = G(x_k)$, writing $G(x_k) = g_{n-1}x_k^{n-1} + \cdots + g_1 x_k + g_0$, we have

$$\begin{aligned}\gamma^r &= g_{n-1}(x_k^r)^{n-1} + \cdots + g_1(x_k^r) + g_0\\ &= g_{n-1}(\alpha^u x_k^v)^{n-1} + \cdots + g_1(\alpha^u x_k^v) + g_0.\end{aligned}$$

Knowing α^u and v, computing γ^r amounts to compute the first n powers of $\alpha^u x_k^v$ and substituting them in G. Since $x_k^n = \alpha$, these powers are all monomials in x_k, and can be computed successively in $O(n)$ multiplications in $\mathbb{F}_q$. Therefore, computing the Frobenius takes a total of $O(n + \log(q))$ operations in $\mathbb{F}_q$.

3.2.4 Trace, norm and quadratic residuosity test

The norm $\mathrm{N}_{\mathbb{L}_k/\mathbb{F}_q}$ and the trace $\mathrm{tr}_{\mathbb{L}_k/\mathbb{F}_q}$ are easy to compute, using transitivity. Indeed, for $\gamma \in \mathbb{L}_k$, we have

$$\mathrm{N}_{\mathbb{L}_k/\mathbb{F}_q}(\gamma) = \mathrm{N}_{\mathbb{L}_1/\mathbb{F}_q}(\mathrm{N}_{\mathbb{L}_2/\mathbb{L}_1}(\cdots \mathrm{N}_{\mathbb{L}_k/\mathbb{L}_{k-1}}(\gamma)))$$

and

$$\mathrm{tr}_{\mathbb{L}_k/\mathbb{F}_q}(\gamma) = \mathrm{tr}_{\mathbb{L}_1/\mathbb{F}_q}(\mathrm{tr}_{\mathbb{L}_2/\mathbb{L}_1}(\cdots \mathrm{tr}_{\mathbb{L}_k/\mathbb{L}_{k-1}}(\gamma))).$$

Write as before $\gamma = G(x_k)$ and $G = G_0(x_{k-1}) + x_k G_1(x_{k-1})$. Then, we have

$$\mathrm{N}_{\mathbb{L}_k/\mathbb{L}_{k-1}}(\gamma) = G_0(x_{k-1})^2 - x_{k-1} G_1(x_{k-1})^2$$

and

$$\mathrm{tr}_{\mathbb{L}_k/\mathbb{L}_{k-1}}(\gamma) = 2G_0(x_{k-1}),$$

since in the quadratic extension $\mathbb{L}_k$ of $\mathbb{L}_{k-1}$ generated by $x_k^2 - x_{k-1}$, the norm (resp. trace) of γ is the product (resp. sum) of γ and its conjugate $\gamma' = G_0(x_{k-1}) - x_k G_1(x_{k-1})$.

To compute the norm of γ, this gives a recursive algorithm using one recursive call and $O(1)$ multiplications in each extension; using the super-linearity of M (as for inversion), the total is $O(\mathsf{M}(n))$ operations in $\mathbb{F}_q$. For the trace, by transitivity, we obtain $\mathrm{tr}_{\mathbb{L}_k/\mathbb{F}_q}(\gamma) = n g_0$ where g_0 is the constant term of G; this could also have been deduced from the fact that the trace of $\gamma = G(x_k)$ in the extension $\mathbb{L}_k = \mathbb{F}_q[X_k]/\langle P_k \rangle$ is the coefficient of X_k^{n-1} in $GP_k' \bmod P_k$. At any rate, the trace is computed using 1 multiplication in $\mathbb{F}_q$

Finally, to check if γ is a quadratic residue we compute $\gamma^{(q^n-1)/2} = \mathrm{N}_{\mathbb{L}_k/\mathbb{F}_q}(\gamma)^{(q-1)/2}$; this takes $O(\mathsf{M}(n) + \log(q))$ operations in $\mathbb{F}_q$ (using repeated squaring for the exponentiation).

Let us briefly comment on alternative derivations of the norm and the trace; they are slightly less efficient, but these ideas will allow us to compute square roots in the next subsection (the underlying idea is not new; it appears for instance in [18, 7]). Fix n and γ as above and for $m \geq 0$, define

$$N_m(\gamma) = \gamma^{1+q+q^2+\cdots+q^{m-1}}$$

to be the *m-norm* of γ. Similarly, the *m-trace* of γ is defined to be $T_m(\gamma) = \gamma + \gamma^q + \cdots + \gamma^{q^{m-1}}$ (this is called a pseudo-trace in [5]). For $m = n$, we recover the standard norm and trace.

Writing

$$\zeta_m = \gamma^{q+q^2+\cdots+q^m},$$

we have $N_m(\gamma) = \gamma\zeta_{m-1}$. The element ζ_m can be computed efficiently by means of the formulas

$$\zeta_1 = \gamma^q \quad \text{and} \quad \zeta_m = \begin{cases} \zeta_{m/2}\zeta_{m/2}^{q^{m/2}} & \text{if } m \text{ is even} \\ \zeta_1\zeta_{m-1}^q & \text{if } m \text{ is odd.} \end{cases} \tag{3.2}$$

(We could compute $N_m(\gamma)$ directly using a similar recurrence, but we will reuse the ζ_m in the next subsection.) Computing ζ_1 is done by means of one Frobenius computation. Deducing ζ_m from either $\zeta_{m/2}$ or $\zeta_{(m-1)/2}$ takes $O(1)$ Frobenius and multiplications. Thus, the total for ζ_m, and thus $N_m(\gamma)$, is $O(\log(m))$ Frobenius and multiplications in $\mathbb{L}_k$, which amounts to $O(\mathsf{M}(n)\log(m) + \log(q)\log(m))$ operations in $\mathbb{F}_q$.

If needed, the m-trace can be computed similarly to the m-norm by the following recursion:

$$T_1 = \gamma \quad \text{and} \quad T_m = \begin{cases} T_{m/2} + T_{m/2}^{q^{m/2}} & \text{if } m \text{ is even} \\ T_1 + T_{m-1}^q & \text{if } m \text{ is odd.} \end{cases}$$

Therefore, $T_m(\gamma)$ can be computed using $O(\log(m))$ Frobenius and additions, hence the overall running time $O(n\log(m) + \log(q)\log(m))$.

3.2.5 Taking square roots

In this subsection, we review the idea presented in [6] to compute square roots, and adapt it to our situation, where computing a Frobenius is cheap.

Let $\delta \in \mathbb{L}_k^\times$ be given, assume that δ is a square and let $\gamma \in \mathbb{L}_k$ be an (unknown) square root of it. Define $\beta \in \mathbb{F}_q$ as the (unknown) quantity

$$\begin{aligned} \beta = \operatorname{tr}_{\mathbb{L}_k/\mathbb{F}_q}(\gamma) = \sum_{i=0}^{n-1} \gamma^{q^i} &= \gamma(1 + \gamma^{q-1} + \gamma^{q^2-1} + \cdots + \gamma^{q^{n-1}-1}) \\ &= \gamma(1 + \delta^{(q-1)/2} + \delta^{(q^2-1)/2} + \cdots + \delta^{(q^{n-1}-1)/2}) \\ &= \gamma\eta, \end{aligned} \tag{3.3}$$

with

$$\eta = 1 + \delta^{(q-1)/2} + \delta^{(q^2-1)/2} + \cdots + \delta^{(q^{n-1}-1)/2}.$$

We may assume $\eta \neq 0$; otherwise, we can replace δ by $\delta' = \delta c^2$ for a random element $c \in \mathbb{L}_k^\times$. We expect to have $\operatorname{tr}_{\mathbb{F}_q/\mathbb{F}_{q'}}(\gamma c) \neq 0$ after $O(1)$ trials: There are q^{2^k}/q values of c for which the trace is zero, and hence the probability of having a non-zero trace is $1 - (q^{2^k}/q)/q^{2^k} = 1 - 1/q \geq 1/2$. Squaring both sides of Eq. (3.3) results in the quadratic equation $\beta^2 = \delta\eta^2$ over $\mathbb{F}_q$.

Provided η is known, β can be deduced from this quadratic equation, and finally γ as $\gamma = \beta\eta^{-1}$. Computing β from the above quadratic equation takes an expected $O(\log(q))$ operations in $\mathbb{F}_q$ [17, Ch. 14.5], so that computing η is the key to computing γ efficiently. This can be done as follows.

Let $\lambda \in \mathbb{L}_k$ be defined by $\lambda = \delta^{(q-1)/2}$; then, η is given by

$$\eta = 1 + \lambda + \lambda^{1+q} + \lambda^{1+q+q^2} + \cdots + \lambda^{1+q+q^2+\cdots+q^{n-2}}.$$

For $m \geq 0$, define

$$\varepsilon_m = \lambda^q + \lambda^{q+q^2} + \cdots + \lambda^{q+q^2+\cdots+q^m},$$

so that $\eta = 1 + \lambda + \lambda\varepsilon_{n-2}$, and recall as well the definition of $\zeta_m = \lambda^{q+q^2+\cdots+q^m}$. Then, similar to the recurrence relation given in (3.2) for ζ, the following holds for ε:

$$\varepsilon_1 = \lambda^q \quad \text{and} \quad \varepsilon_m = \begin{cases} \varepsilon_{m/2} + \zeta_{m/2}\varepsilon_{m/2}^{q^{m/2}} & \text{if } m \text{ is even} \\ \varepsilon_{m-1} + \zeta_m & \text{if } m \text{ is odd} \end{cases} \tag{3.4}$$

Computing λ takes $O(\mathsf{M}(n)\log(q))$ operations in $\mathbb{F}_q$; then, we obtain the initial values ζ_1 and ε_1 using $O(1)$ Frobenius. Assume, inductively, that we have computed ε_m, ζ_m. Then, using Eqs. (3.2) and (3.4), ζ_{2m} and ε_{2m}, or ζ_{2m+1} and ε_{2m+1}, can be computed using $O(1)$ Frobenius and $O(1)$ multiplications in $\mathbb{F}_{q^n}$. Therefore, ε_n, and altogether $\eta = 1 + \lambda + \lambda\varepsilon_{n-2}$ can be computed using $O(\mathsf{M}(n)\log(q) + \mathsf{M}(n)\log(n)) = O(\mathsf{M}(n)\log(nq))$ operations in $\mathbb{F}_q$. We have seen that deducing β takes an expected $O(\log(q))$ operations in $\mathbb{F}_q$, and the cost of computing γ is negligible compared to the computation of η. Thus, the overall running time is an expected $O(\mathsf{M}(n)\log(nq))$ operations in $\mathbb{F}_q$.

3.2.6 Computing embeddings

We finally consider the problem of computing embeddings and isomorphisms between two different "towers" defining the quadratic closure of $\mathbb{F}_q$. Consider $\mathbb{L}_k = \mathbb{F}_q[X_k]/\langle P_k\rangle$ and $\mathbb{L}'_j = \mathbb{F}_q[Y_j]/\langle Q_j\rangle$, k, j positive integers, where we write

$$P_k = X_k^{2^k} - \alpha \quad \text{and} \quad Q_j = Y_j^{2^j} - \beta,$$

for some non-quadratic residues α, β in $\mathbb{F}_q$. Assuming for instance that $k \leq j$, $\mathbb{L}_k$ can be identified as a subfield of $\mathbb{L}'_j$; we show how to compute an embedding $\phi : \mathbb{L}_k \hookrightarrow \mathbb{L}'_j$ efficiently. As before, we denote by x_k the image of X_k in $\mathbb{L}_k$, and by y_j the image of Y_j in $\mathbb{L}'_j$; we write $n = 2^k$ and $m = 2^j$.

The idea is straightforward: we first find a root ρ of P_k in $\mathbb{L}'_j$; then, the mapping $\phi : \mathbb{L}_k \to \mathbb{L}'_j$ given by $\phi(G(x_k)) = G(\rho) \bmod Q_j$ is well-defined and gives an isomorphism of $\mathbb{L}_k$ onto its image. To find the root ρ of P_k in $\mathbb{L}'_j$, one has to take k successive square roots of α in $\mathbb{L}'_j$. For this, we could use the algorithm of the previous subsection, but since we start from $\alpha \in \mathbb{F}_q$, better can be done.

Let us look at a slightly more general question: given $\mu \in \mathbb{F}_q$ and an even integer $\ell \geq 0$, compute a square root of μy_j^ℓ:

- if μ is a square in $\mathbb{F}_q$, say $\mu = \nu^2$, $\nu y_j^{\ell/2}$ is a square root of μy_j^ℓ;
- else, μ/β is a square in $\mathbb{F}_q$, say $\mu/\beta = \nu^2$; then $\nu y_j^{2^{j-1}+\ell/2}$ is a square root of μy_j^ℓ.

Since we start with $\ell = 0$, we can repeat the process at least j times, and thus in particular at least k times; the cost is thus that of $O(k) = O(\log(n))$ quadratic residuosity tests, square-root computations and arithmetic operations in $\mathbb{F}_q$; this uses an expected $O(\log(q)\log(n))$ operations in $\mathbb{F}_q$. Since the root ρ we obtain is of the form $\rho = \mu y_j^\ell$ for some $\ell < m$, computing $G(\rho) \bmod Q_j$, for some G of degree less than n (and thus than m), takes $O(m)$ multiplications in $\mathbb{F}_q$ (as was the case for Frobenius computation).

For isomorphism computation, taking $k = j$, and thus $m = n$, gives the claimed bound $O(n + \log(q)\log(n))$ operations in $\mathbb{F}_q$.

3.3 Experiments

We conclude this section with experiments using an implementation of our algorithms based on NTL [14]. All running times are obtained on an Intel Xeon CPU. In all cases, we start from the base field $\mathbb{F}_p$.

First, we consider square root computation. For computing square roots in an extension $\mathbb{F}_{p^n}$, without assumption on n, we used in [6] modular polynomial composition, resulting in the running time $O(\mathsf{M}(n)\log(p) + \mathsf{C}(n)\log(n))$. In cases where n is a power of two, the results in this paper are superior in terms of complexity; Figure 3.1 confirms that this is also the case in practice. In that figure, we take the "random" prime $p = 348975609381470925634534573457497$ already used in [6], and different values of the extension degree n, that are always powers of two; see below for the reasons behind our choice of such a large value of p.

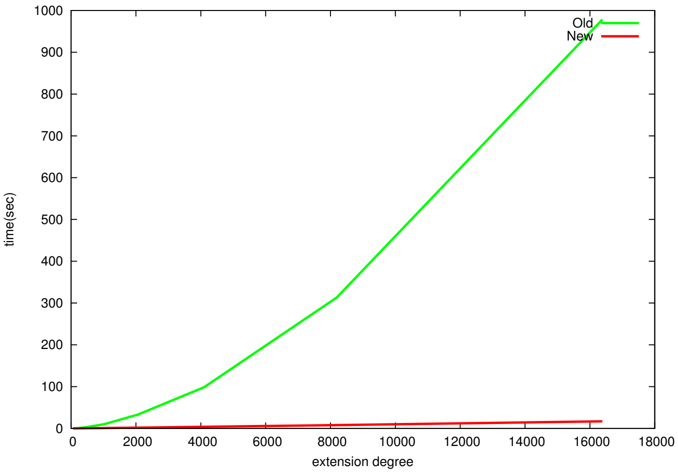


Figure 3.1: The new square root algorithm vs. the one in [6]

Table 3.2 gives timings (in seconds) for the genus 2 point-counting application described in the introduction. The table describes the various ingredients involved in computing successive 2^k-torsion divisors in the Jacobian of the (randomly chosen) curve C:

$$\begin{aligned} y^2 \quad = \quad & x^5 + 674123654726631691190853807697321377727\, x^4 \\ & +132706051439871719391705031627238584248\, x^3 \\ & +150906984006321211274278480789580538770\, x^2 \\ & +560222282607748278280534730775992622 4\, x \\ & +157456212652423046465778673243920804193. \end{aligned}$$

over $\mathbb{F}_p$ with $p = 2^{127} - 1$ (this 127 bit prime is the one used in the records described in [8]). Once these divisors are known, they are used to compute the cardinality of the Jacobian J of C modulo 2^k (more exactly, we compute the image modulo 2^k of the characteristic polynomial of the Frobenius endomorphism on J); this last step is not detailed here, we will refer to it as the *search* step.

In Table 3.2, the degree e_k is the degree of the field extension over which the successive divisors are defined. There are two main rows: the first one gives the timings for computing all required square roots (which give us the required 2^k-torsion divisors); the second row gives the timing for the search step, which involves arithmetic operations in the current extension of $\mathbb{F}_p$. For a more precise profiling, each of the two main rows is divided into three subrows labelled with I, II, and III: I denotes the original Gaudry and Schost implementation [8]; II and III denote the same implementation but using the algorithm in [6] and the one in Section 3.2.5 respectively. All square root algorithms in this table are probabilistic.

In previous implementations, square root computation was a clear bottleneck; with our new algorithm, it has now become a minor component of the running time.

index 2^k		2^6	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}
degree e_k		2^5	2^6	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}
square roots	I	0.2	0.4	1.2	3.5	11	33	109	365	1262	4466	16246	60689
	II	0.2	0.5	1.2	2.9	8	23	73	232	734	2309	7368	23604
	III	0.1	0.2	0.5	1.1	2	5	11	25	53	114	246	523
search step	I	0.5	1.1	2.8	6.5	14	32	73	164	368	816	2020	4827
	II	0.4	1.0	2.3	5.4	12	27	62	139	309	657	1609	3740
	III	0.4	0.9	2.0	4.5	11	24	53	119	267	598	1402	3297

Table 3.2: Timings for lifting 2^k-torsion

Bibliography

[1] A. Bostan, M. F. I. Chowdhury, J. van der Hoeven, and É. Schost. Homotopy methods for multiplication modulo triangular sets. *Journal of Symbolic Computation*, 46(12):1378–1402, 2011.

[2] R. P. Brent and H. T. Kung. Fast algorithms for manipulating formal power series. *Journal of the Association for Computing Machinery*, 25(4):581–595, 1978.

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

[4] M. Cipolla. Un metodo per la risoluzione della congruenza di secondo grado. *Napoli Rend.*, 9:153–163, 1903.

[5] L. De Feo and É. Schost. Fast arithmetics in Artin-Schreier towers over finite fields. *Journal of Symbolic Computation*, 47(7):771–792, 2012.

[6] J. Doliskani and É. Schost. Taking roots over high extensions of finite fields. *Mathematics of Computation*, 2012. To appear.

[7] W. Feng, Y. Nogami, and Y. Morikawa. A fast square root computation using the Frobenius mapping. In *Information and Communications Security*, volume 2836 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2003.

[8] P. Gaudry and É. Schost. Genus 2 point counting over prime fields. *Journal of Symbolic Computation*, 47(4):368 – 400, 2012.

[9] E. Kaltofen and V. Shoup. Fast polynomial factorization over high algebraic extensions of finite fields. In *ISSAC'97*, pages 184–188. ACM, 1997.

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

[11] S. Lang. *Algebra*, volume 211 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, third edition, 2002.

[12] R. Schoof. Elliptic curves over finite fields and the computation of square roots mod p. *Mathematics of Computation*, 44:483–494, 1985.

[13] D. Shanks. Five number-theoretic algorithms. In *Proceedings of the Second Manitoba Conference on Numerical Mathematics*, pages 51–70, 1972.

[14] V. Shoup. A library for doing number theory (NTL). `http://www.shoup.net/ntl/`.

[15] V. Shoup. Fast construction of irreducible polynomials over finite fields. *Journal of Symbolic Computation*, 17(5):371–391, 1994.

[16] A. Tonelli. Bemerkung über die Auflösung quadratischer Congruenzen. *Göttinger Nachrichten*, pages 344–346, 1891.

[17] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, second edition, 2003.

[18] J. von zur Gathen and V. Shoup. Computing Frobenius maps and factoring polynomials. *Comput. Complexity*, 2(3):187–224, 1992.

[19] Feng Wang, Yasuyuki Nogami, and Yoshitaka Morikawa. An efficient square root computation in finite fields $GF(p^{2^d})$. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, E88-A(10):2792–2799, October 2005.

Chapter 4

Fast Algorithms for ℓ-adic Towers over Finite Fields

4.1 Introduction

Building arbitrary finite extensions of finite fields is a fundamental task in any computer algebra system. For this, an especially powerful system is the "compatibly embedded finite fields" implemented in Magma [2, 3], capable of building extensions of any finite field and keeping track of the embeddings between the fields.

The system described in [3] uses linear algebra to describe the embeddings of finite fields. From a complexity point of view, this is far from optimal: one may hope to compute and apply the morphisms in quasi-linear time in the degree of the extension, but this is usually out of reach of linear algebra techniques. Even worse, the quadratic memory requirements make the system unsuitable for embeddings of large degree extensions. Although the Magma core has evolved since the publication of the paper, experiments in Section 4.5 show that embeddings of large extension fields are still out of reach.

In this paper, we discuss an approach based on polynomial arithmetic, rather than linear algebra, with much better performance. We consider here one aspect of the question, ℓ-adic towers; we expect that this will play an important role towards a complete solution.

Let q be a power of a prime p, let $\mathbb{F}_q$ be the finite field with q elements and let ℓ be a prime. Our main interest in this paper is on the algorithmic aspects of the *ℓ-adic closure* of $\mathbb{F}_q$, which is defined as follows. Fix arbitrary embeddings

$$\mathbb{F}_q \subset \mathbb{F}_{q^\ell} \subset \mathbb{F}_{q^{\ell^2}} \subset \cdots ;$$

then, the ℓ-adic closure of $\mathbb{F}_q$ is the infinite field defined as

$$\mathbb{F}_q^{(\ell)} = \bigcup_{i \geq 0} \mathbb{F}_{q^{\ell^i}}.$$

We also call an *ℓ-adic tower* the sequence of extensions $\mathbb{F}_q, \mathbb{F}_{q^\ell}, \ldots$ In particular, they allow us to build the algebraic closure $\bar{\mathbb{F}}_q$ of $\mathbb{F}_q$, as there is an isomorphism

$$\bar{\mathbb{F}}_q \cong \bigotimes_{\ell \text{ prime}} \mathbb{F}_q^{(\ell)}, \tag{4.1}$$

where the tensor products are over $\mathbb{F}_q$; we will briefly mention below the algorithmic counterpart of this equality.

We present here algorithms that allow us to "compute" in the first levels of ℓ-adic towers (in a sense defined hereafter); at level i, our goal is to be able to perform all basic operations in quasi-linear time in the extension degree ℓ^i. We do not discuss the representation of the base field $\mathbb{F}_q$, and we count operations $\{+,-,\times,\div\}$ in $\mathbb{F}_q$ at unit cost.

Our techniques are inspired by those in [4, 5, 8], which dealt with the Artin-Schreier case $\ell = p$ (see also [9], which reused these ideas in the case $\ell = 2$): we construct families of irreducible polynomials with special properties, then give algorithms that exploit the special form of those polynomials to apply the embeddings. Because they are treated in the references [8, 9], *we exclude the cases $\ell = p$ and $\ell = 2$.*

The field $\mathbb{F}_{q^{\ell^i}}$ will be represented as $\mathbb{F}_q[X_i]/\langle Q_i\rangle$, for some irreducible polynomial $Q_i \in \mathbb{F}_q[X_i]$. Letting x_i be the residue class of X_i modulo Q_i endows $\mathbb{F}_{q^{\ell^i}}$ with the monomial basis

$$\mathbf{U}_i = (1, x_i, x_i^2, \ldots, x_i^{\ell^i-1}). \tag{4.2}$$

Let $\mathsf{M} : \mathbb{N} \to \mathbb{N}$ be such that polynomials in $\mathbb{F}_q[X]$ of degree less than n can be multiplied in $\mathsf{M}(n)$ operations in $\mathbb{F}_q$, under the assumptions of [33, Ch. 8.3]; using FFT multiplication, one can take $\mathsf{M}(n) \in O(n \log(n) \log\log(n))$. Then, multiplications and inversions in $\mathbb{F}_q[X_i]/\langle Q_i\rangle$ can be done in respectively $O(\mathsf{M}(\ell^i))$ and $O(\mathsf{M}(\ell^i)\log(\ell^i))$ operations in $\mathbb{F}_q$ [33, Ch. 9-11]. This is almost optimal, as both results are quasi-linear in $[\mathbb{F}_{q^{\ell^i}} : \mathbb{F}_q] = \ell^i$.

Computing embeddings requires more work. For this problem, it is enough consider a pair of consecutive levels in the tower, as any other embedding can be done by applying repeatedly this elementary operation. Following again [8], we introduce two slightly more general operations, *lift* and *push*.

To motivate them, remark that for $i \geq 2$, $\mathbb{F}_{q^{\ell^i}}$ has two natural bases as a vector space over $\mathbb{F}_q$. The first one is via the monomial basis $\mathbf{U}_i$ seen above, corresponding to the univariate model $\mathbb{F}_q[X_i]/\langle Q_i\rangle$. The second one amounts to seeing $\mathbb{F}_{q^{\ell^i}}$ as a degree ℓ extension of $\mathbb{F}_{q^{\ell^{i-1}}}$, that is, as

$$\mathbb{F}_q[X_{i-1}, X_i]/\langle Q_{i-1}(X_{i-1}), T_i(X_{i-1}, X_i)\rangle, \tag{4.3}$$

for some polynomial T_i monic of degree ℓ in X_i, and of degree less than ℓ^{i-1} in X_{i-1}. The corresponding basis is bivariate and involves x_{i-1} and x_i:

$$\mathbf{B}_i = (1, \ldots, x_{i-1}^{\ell^{i-1}-1}, \ldots, x_i^{\ell-1}, \ldots, x_{i-1}^{\ell^{i-1}-1} x_i^{\ell-1}). \tag{4.4}$$

Lifting corresponds to the change of basis from $\mathbf{B}_i$ to $\mathbf{U}_i$; *pushing* is the inverse transformation.

Lift and push allow us to perform embeddings as a particular case, but they are also the key to many further operations. We do not give details here, but we refer the reader to [8, 9, 18] for examples such as the computation of relative traces, norms or characteristic polynomials, and applications to solving Artin-Schreier or quadratic equations, given in [8] and [9] for respectively $\ell = p$ and $\ell = 2$.

Table 4.1 summarizes our main results. Under various assumptions, it gives costs (counted in terms of operations in $\mathbb{F}_q$) for initializing the construction, building the polynomials Q_i and

Condition	Initialization	Q_i, T_i	Lift, push
$q = 1 \bmod \ell$	$O_e(\log(q))$	$O(\ell^i)$	$O(\ell^i)$
$q = -1 \bmod \ell$	$O_e(\log(q))$	$O(\ell^i)$	$O(\mathsf{M}(\ell^i)\log(\ell^i))$
$-$	$O_e(\ell^2 + \mathsf{M}(\ell)\log(q))$	$O(\mathsf{M}(\ell^{i+1})\mathsf{M}(\ell)\log(\ell^i)^2)$	$O(\mathsf{M}(\ell^{i+1})\mathsf{M}(\ell)\log(\ell^i))$
$4\ell \leq q^{1/4}$	$\tilde{O_e}(\ell\log^5(q) + \ell^3)$ (bit)	$O_e(\ell^2 + \mathsf{M}(\ell)\log(\ell q) + \mathsf{M}(\ell^i)\log(\ell^i))$	$O(\mathsf{M}(\ell^i)\log(\ell^i))$
$4\ell \leq q^{1/4}$	$\tilde{O_e}(\ell\log^5(q))$ (bit) $+ O_e(\mathsf{M}(\ell)\sqrt{q}\log(q))$	$O_e(\log(q) + \mathsf{M}(\ell^i)\log(\ell^i))$	$O(\mathsf{M}(\ell^i)\log(\ell^i))$

Table 4.1: Summary of results

T_i from Eq.(4.3), and performing lift and push. $O_e(\ )$ indicates probabilistic algorithms with expected running time, and $\tilde{O_e}(\ )$ indicates the additional omission of logarithmic factors. Two entries mention bit complexity, as they use an elliptic curve point counting algorithm.

In all cases, our results are close to being linear-time in ℓ^i, up to sometimes the loss of a factor polynomial in ℓ. Except for the (very simple) case where $q = 1 \bmod \ell$, these results are new, to the best of our knowledge. To otbain them, we use two constructions: the first one (Section 4.2) uses cyclotomy and descent algorithms; the second one (Section 4.3) relies on the construction of a sequence of fibers of isogenies between algebraic groups.

These constructions are inspired by previous work due to respectively Shoup [27, 28] and Lenstra / De Smit [21], and Couveignes / Lercier [6]. We briefly discuss them here and give more details in the further sections.

Lenstra and De Smit [21] address a question similar to ours, the construction of the ℓ-adic closure of $\mathbb{F}_q$ (and of its algebraic closure), with the purpose of standardizing it. The resulting algorithms run in polynomial time, but (implicitly) rely on linear algebra and multiplication tables, so quasi-linear time is not directly reachable. References [27, 28, 6] discuss a related problem, the construction of irreducible polynomials over $\mathbb{F}_q$; the question of computing embeddings is not considered. The results in [6] are *quasi-linear*, but they rely on an algorithm by Kedlaya and Umans [14] that works only in a boolean model.

To conclude the introduction, let us mention a few applications of our results. A variety of computations in number theory and algebraic geometry require constructing new extension fields and moving elements from one to the other. As it turns out, in many cases, the ℓ-adic constructions considered here are sufficient: two examples are [7, 11], both in relation to torsion subgroups of Jacobians of curves.

The main question remains of course the cost of computing in arbitrary extensions. As showed by Eq. (4.1), this boils down to the study of ℓ-adic towers, as done in this paper, together with algorithms for computing in *composita*. References [27, 28, 6] deal with related questions for the problem of computing irreducible polynomials; a natural follow-up to the present work is to study the cost of embeddings and similar changes of bases in this more general context.

4.2 Quasi-cyclotomic towers

In this section, we discuss a construction of the ℓ-adic tower over $\mathbb{F}_q$ inspired by previous work of Shoup [27, 28], Lenstra-De Smit [21] and Couveignes-Lercier [6]. The results of this section establish rows 1 and 3 of Table 4.1.

The construction starts by building an extension $\mathbb{K}_0 = \mathbb{F}_q[Y_0]/\langle P_0\rangle$ obtained by adjoining an ℓth

root of unity to $\mathbb{F}_q$, such that the residue class y_0 of Y_0 is a non ℓ-adic residue in $\mathbb{K}_0$ (we discuss this in more detail in the first subsection); we let r be the degree of P_0. By [17, Th. VI.9.1], for $i \geq 1$, the polynomial $Y_i^{\ell^i} - y_0 \in \mathbb{K}_0[Y_i]$ is irreducible, so $\mathbb{K}_i = \mathbb{K}_0[Y_i]/\langle Y_i^{\ell^i} - y_0 \rangle$ is a field with $q^{r\ell^i}$ elements. Letting y_i be the residue class of Y_i in $\mathbb{K}_i$, these fields are naturally embedded in one another by the isomorphism $\mathbb{K}_{i+1} \simeq \mathbb{K}_i[Y_{i+1}]/\langle Y_{i+1}^{\ell} - y_i \rangle$; in particular, we have $y_{i+1}^{\ell} = y_i$.
In order to build $\mathbb{F}_{q^{\ell^i}}$, we apply a descent process, for which we follow an idea of Shoup's. For $i \geq 0$, let x_i be the trace of y_i over a subfield of index r:

$$x_i \;=\; \textstyle\sum_{0 \leq j \leq r-1} y_i^{q^{\ell^i j}}.$$

Then, [27, Th. 2.1] proves that $\mathbb{F}_q(x_i) = \mathbb{F}_{q^{\ell^i}}$ (see Figure 4.1). In particular, the minimal polynomials of $x_1, x_2, \dots$ over $\mathbb{F}_q$ are the irreducible polynomials Q_i we are interested in.

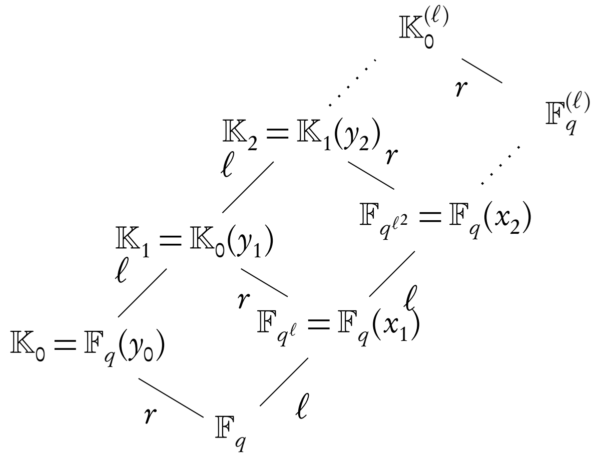

$\mathbb{K}_0^{(\ell)}$
r
$\mathbb{F}_q^{(\ell)}$
$\mathbb{K}_2 = \mathbb{K}_1(y_2)$
ℓ
r
$\mathbb{F}_{q^{\ell^2}} = \mathbb{F}_q(x_2)$
$\mathbb{K}_1 = \mathbb{K}_0(y_1)$
ℓ
r
ℓ
$\mathbb{F}_{q^\ell} = \mathbb{F}_q(x_1)$
$\mathbb{K}_0 = \mathbb{F}_q(y_0)$
r
$\mathbb{F}_q$
ℓ

Figure 4.1: The ℓ-adic towers over $\mathbb{F}_q$ and $\mathbb{K}_0$.

We show here how to compute these polynomials, the polynomials T_i of Eq. (4.3) and how to perform lift and push. To this effect, we will define more general minimal polynomials: for $0 \leq j \leq i$, we will let $Q_{i,j} \in \mathbb{F}_q(x_j)[X_i]$ be the minimal polynomial of x_i over $\mathbb{F}_q(x_j)$, so that $Q_{i,j}$ has degree ℓ^{i-j}, with in particular $Q_{i,0} = Q_i$ and $Q_{i,i-1} = T_i(x_{i-1}, X_i)$.
In Subsections 4.2.2 and 4.2.3, we discuss favorable cases, where ℓ divides respectively $q-1$ and $q+1$. The first case is folklore; it yields the fastest and simplest algorithms. Our results for the second case are related to known facts about Chebyshev polynomials [30, § 6.2], but, to the best of our knowledge, are new. We will revisit these cases in Section 4.3 and account for their naming convention. Our results in the general case (Subsection 4.2.4) are slower, but still quasi-linear in ℓ^i, up to a factor polynomial in ℓ.
Shoup used this setup to compute Q_i in time quadratic in ℓ^i [28, Th. 11]. It is noted there that using *modular composition* techniques [33, Ch. 12], this could be improved to get a subquadratic exponent in ℓ^i, up to an extra cost polynomial in ℓ. For $\ell = 3$ (where we are in one the first two cases), Couveignes and Lercier make a similar remark in [6, § 2.4]; using a result by Kedlaya and Umans [14] for modular composition, they derive for any $\varepsilon > 0$ a cost of $3^{i(1+\varepsilon)} O(\log(q))$ *bit* operations, up to polynomial terms in $\log\log(q)$.

In this section, and in the rest of this paper, if L/K is a field extension, we write $\mathrm{tr}_{L/K}$, $\mathrm{N}_{L/K}$ and $\mathrm{Gal}_{L/K}$ for the trace, norm and Galois group of the extension.

4.2.1 Finding P_0

To determine P_0, we compute the ℓ-th cyclotomic polynomial $\Phi_\ell \in \mathbb{Z}[X_0]$ and factor it over $\mathbb{F}_q[X_0]$: by [28, Th. 9], this takes $O_e(\mathsf{M}(\ell)\log(\ell q))$ operations in $\mathbb{F}_q$.

Over $\mathbb{F}_q[X_0]$, Φ_ℓ splits into irreducible factors of the same degree r, where r is the order of q in $\mathbb{Z}/\ell\mathbb{Z}$ (so r divides $\ell-1$); let F_0 be one of these factors. By construction, there exist non ℓ-adic residues in $\mathbb{F}_q[X_0]/\langle F_0\rangle$. Once such a non-residue y_0 is found, we simply let P_0 be its minimal polynomial over $\mathbb{F}_q$ (which still has degree r); given y_0, computing P_0 takes $O(r^2)$ operations in $\mathbb{F}_q$ [28, Th. 4].

Following [27, 28, 6], we pick y_0 at random: we expect to find a non-residue after $O(1)$ trials; by [28, Lemma 15], each takes $O_e(\mathsf{M}(\ell)\log(r)+\mathsf{M}(r)\log(\ell)\log(r)+\mathsf{M}(r)\log(q))$ operations in $\mathbb{F}_q$. An alternative due to Lenstra and De Smit is to take iterated ℓ-th roots of $X_0 \bmod F_0$ until we find a non-residue: this idea is helpful in making the construction canonical, but more costly, so we do not consider it.

4.2.2 $\mathbb{G}_m$-type extensions

We consider here the simplest case, where ℓ divides $q-1$; the (classical) facts below give the first row of Table 4.1. In this case, Φ_ℓ splits into linear factors over $\mathbb{F}_q$ (so $r=1$). The polynomial P_0 is of the form Y_0-y_0, where y_0 is a non ℓ-adic residue in $\mathbb{F}_q$; since we can bypass the factorization of Φ_ℓ, the cost of initialization is $O_e(\log(q))$ operations in $\mathbb{F}_q$. Besides, no descent is required: for $i \ge 0$, we have $\mathbb{K}_i = \mathbb{F}_{q^{\ell^i}}$ and $x_i = y_i$; the families of polynomials we obtain are

$$Q_i = X_i^{\ell^i} - y_0 \quad \text{and} \quad T_i = X_i^{\ell} - X_{i-1}. \tag{4.5}$$

Lift and push use no operation in $\mathbb{F}_q$, only exponent arithmetic. Lift takes $F=\sum_{0\le j<\ell^{i+1}} f_j x_{i+1}^j$ and rewrites it as a bivariate polynomial in x_i, x_{i+1} and push does the converse operation, using the rules

$$x_{i+1}^j = x_i^{j \text{ div } \ell} x_{i+1}^{j \bmod \ell} \quad \text{and} \quad x_i^e x_{i+1}^f = x_{i+1}^{e\ell+f}.$$

4.2.3 Chebyshev-type extensions

Consider now the case where ℓ divides $q+1$: then, Φ_ℓ splits into quadratic factors over $\mathbb{F}_q$ and $r=2$. We also require that y_0 has norm 1 over $\mathbb{F}_q$ (see below for a discussion); we deduce an expression for the polynomials $Q_{i,j} \in \mathbb{F}_q(x_j)[X_i]$.

Proposition 5. *For $1 \le j < i$, $Q_{i,j}$ satisfies*

$$Q_{i,j}(X_i) = Y^{\ell^{i-j}} + Y^{-\ell^{i-j}} - x_j \mod Y^2 - X_i Y + 1. \tag{4.6}$$

Proof. Since $\mathrm{N}_{\mathbb{K}_0/\mathbb{F}_q}(y_0) = 1$, $\mathrm{N}_{\mathbb{K}_i/\mathbb{F}_q(x_i)}(y_i)$ is an ℓ^i-th root of unity. But ℓ does not divide $q-1$, so 1 is the only such root in $\mathbb{F}_q$, and by induction on i it also is the only root in $\mathbb{F}_q(x_i)$; hence, the minimal polynomial of y_i over $\mathbb{F}_q(x_i)$ is $Y_i^2 - x_i Y_i + 1$. By composition, it follows that the minimal polynomial of y_i over $\mathbb{F}_q(x_j)$ is $Y_i^{2\ell^{i-j}} - x_j Y_i^{\ell^{i-j}} + 1$. Taking a resultant to eliminate Y_i between these two polynomials gives the following relation between x_j and x_i:

$$Q_{i,j}(X_i)^2 = \mathrm{Res}_{Y_i}(Y_i^{2\ell^{i-j}} - x_j Y_i^{\ell^{i-j}} + 1,\ Y_i^2 - X_i Y_i + 1).$$

By direct calculation, this is equivalent to Eq. (4.6). □

As a result, we can compute $Q_{i,j}$ in time $O(\mathsf{M}(\ell^{i-j}))$ by repeated squaring, but we give a better algorithm in Section 4.3.1 (and show how to find a y_0 satisfying the hypotheses); we leave the algorithms for lift and push to Section 4.4.

4.2.4 The general case

Finally, we discuss the general case, with no assumption on the behavior of Φ_ℓ in $\mathbb{F}_q[X]$. This completes the third row of Table 4.1, using the bound $r \in O(\ell)$. Because $r = [\mathbb{K}_0 : \mathbb{F}_q]$ divides $\ell - 1$, it is coprime with ℓ. Thus, Q_i remains the minimal polynomial of x_i over $\mathbb{K}_0$, and more generally $Q_{i,j}$ remains the minimal polynomial of x_i over $\mathbb{K}_j$; this will allow us to replace $\mathbb{F}_q$ by $\mathbb{K}_0$ as our base field. We will measure all costs by counting operations in $\mathbb{K}_0$, and we will deduce the cost over $\mathbb{F}_q$ by adding a factor $O(\mathsf{M}(r)\log(r))$ to account for the cost of arithmetic in $\mathbb{K}_0$.

For $i \geq 0$, since $\mathbb{K}_i = \mathbb{K}_0[Y_i]/\langle Y_i^{\ell^i} - y_0 \rangle$, we represent its elements on the basis $\{y_i^e \mid 0 \leq e < \ell^i\}$; e.g., x_i is written as

$$x_i \;=\; \textstyle\sum_{0 \leq j \leq r-1} y_i^{q^{\ell^i j} \bmod \ell^i} y_0^{q^{\ell^i j} \operatorname{div} \ell^i}.$$

Our strategy is to convert between two univariate bases of $\mathbb{K}_i$, $\{y_i^e \mid 0 \leq e < \ell^i\}$ and $\{x_i^e \mid 0 \leq e < \ell^i\}$. In other words, we show how to apply the isomorphism

$$\Psi_i : \mathbb{K}_i = \mathbb{K}_0[Y_i]/\langle Y_i^{\ell^i} - y_0 \rangle \to \mathbb{K}_0[X_i]/\langle Q_{i,0} \rangle$$

and its inverse; we will compute the required polynomials $Q_{i,0}$ and $Q_{i,i-1}$ as a byproduct. In a second time, we will use Ψ_i to perform push and lift between the monomial basis in x_i and the bivariate basis in (x_{i-1}, x_i).

We will factor Ψ_i into elementary isomorphisms

$$\Psi_{i,j} : \mathbb{K}_j[X_i]/\langle Q_{i,j} \rangle \to \mathbb{K}_{j-1}[X_i]/\langle Q_{i,j-1} \rangle, \quad j = i, \ldots, 1.$$

To start the process, with $j = i$, we let $Q_{i,i} = X_i - x_i \in \mathbb{K}_i[X_i]$, so that $\mathbb{K}_i = \mathbb{K}_i[X_i]/\langle Q_{i,i} \rangle$. Take now $j \leq i$ and suppose that $Q_{i,j}$ is known. We are going to factor $\Psi_{i,j}$ further as $\Phi''_{i,j} \circ \Phi'_{i,j} \circ \Phi_{i,j}$, by introducing first the isomorphism

$$\varphi_j : \mathbb{K}_j \to \mathbb{K}_{j-1}[Y_j]/\langle Y_j^\ell - y_{j-1} \rangle.$$

The forward direction is a push from the monomial basis in y_j to the bivariate basis in (y_{j-1}, y_j) and the inverse is a lift; none of them involves any arithmetic operation (see Subsection 4.2.2). Then, we deduce the isomorphism

$$\Phi_{i,j} : \mathbb{K}_j[X_i]/\langle Q_{i,j}\rangle \to \mathbb{K}_{j-1}[Y_j, X_i]/\langle Y_j^\ell - y_{j-1}, Q^\star_{i,j}\rangle,$$

where $Q^\star_{i,j}$ is obtained by applying φ_j to all coefficients of $Q_{i,j}$. Since $\Phi_{i,j}$ consists in a coefficient-wise application of φ_j, applying it or its inverse costs no arithmetic operations.

Next, changing the order of Y_j and X_i, we deduce that there exists $S_{i,j}$ in $\mathbb{K}_{j-1}[X_j]$ and an isomorphism

$$\Phi'_{i,j} : \mathbb{K}_{j-1}[Y_j, X_i]/\langle Y_j^\ell - y_{j-1}, Q^\star_{i,j}\rangle \to \mathbb{K}_{j-1}[X_i, Y_j]/\langle Q_{i,j-1}, Y_j - S_{i,j}\rangle,$$

where $\deg(Q^\star_{i,j}, X_i) = \ell^{i-j}$ and $\deg(Q_{i,j-1}, X_i) = \ell^{i-j+1}$.

Lemma 6. *From $Q^\star_{i,j}$, we can compute $Q_{i,j-1}$ and $S_{i,j}$ in $O(\mathsf{M}(\ell^{i+1})\log(\ell^i))$ operations in $\mathbb{K}_0$. Once this is done, we can apply $\Phi'_{i,j}$ or its inverse in $O(\mathsf{M}(\ell^{i+1}))$ operations in $\mathbb{K}_0$.*

Proof. We obtain $Q_{i,j-1}$ and $S_{i,j}$ from the resultant and degree-1 subresultant of $Y_j^\ell - y_{j-1}$ and $Q^\star_{i,j}$ with respect to Y_j, computed over the polynomial ring $\mathbb{K}_{j-1}[X_i]$. This is done by the algorithms of [24, 22], using $O(\mathsf{M}(\ell^{i+1})\log(\ell))$ operations in $\mathbb{K}_0$ (for this analysis, and all others in this proof, we assume that we use Kronecker's substitution for multiplications). To obtain $S_{i,j}$, we invert the leading coefficient of the degree-1 subresultant modulo the resultant $Q_{i,j-1}$; this takes $O(\mathsf{M}(\ell^i)\log(\ell^i))$ operations in $\mathbb{K}_0$.

Applying $\Phi'_{i,j}$ amounts to taking a polynomial $A(Y_j, X_i)$ reduced modulo $\langle Y_j^\ell - y_{j-1}, Q^\star_{i,j}\rangle$ and reducing it modulo $\langle Q_{i,j-1}, Y_j - S_{i,j}\rangle$. This is done by computing $A(S_{i,j}, X_i)$, doing all operations modulo $Q_{i,j-1}$. Using Horner's scheme, this takes $O(\ell)$ operations $(+, \times)$ in $\mathbb{K}_{j-1}[X_i]/\langle Q_{i,j-1}\rangle$, so the complexity claim follows.

Conversely, we start from $A(X_i)$ reduced modulo $Q_{i,j-1}$; we have to reduce it modulo $\langle Y_j^\ell - y_{j-1}, Q^\star_{i,j}\rangle$. This is done using the fast Euclidean division algorithm with coefficients in $\mathbb{K}_{j-1}[Y_j]/\langle Y_j^\ell - y_{j-1}\rangle$ for $O(\mathsf{M}(\ell^{i+1}))$ operations in $\mathbb{K}_0$. □

The last isomorphism $\Phi''_{i,j}$ is trivial:

$$\Phi''_{i,j} : \mathbb{K}_{j-1}[X_i, Y_j]/\langle Q_{i,j-1}, Y_j - S_{i,j}\rangle \to \mathbb{K}_{j-1}[X_i]/\langle Q_{i,j-1}\rangle$$

forgets the variable Y_j; it requires no arithmetic operation.

Taking $j = i, \ldots, 1$ allows us to compute $Q_{i,i-1}$ and $Q_{i,0}$ for $O(i^2\mathsf{M}(\ell^{i+1})\log(\ell))$ operations in $\mathbb{K}_0$. Composing the maps $\Psi_{i,j}$, we deduce further that we can apply Ψ_i or its inverse for $O(i\mathsf{M}(\ell^{i+1}))$ operations in $\mathbb{K}_0$.

We claim that we can then perform push and lift between the monomial basis in x_i and the bivariate basis in (x_{i-1}, x_i) for the same cost. Let us for instance explain how to lift.

We start from A written on the bivariate basis in (x_{i-1}, x_i); that is, A is in $\mathbb{K}_0[X_{i-1}, X_i]/\langle Q_{i-1}, T_i\rangle$. Apply Ψ_{i-1} to its coefficients in $x_i^0, \ldots, x_i^{\ell-1}$, to rewrite A as an element of

$$\mathbb{K}_0[Y_{i-1}, X_i]/\langle Y_{i-1}^{\ell^{i-1}} - y_{i-2}, T_i\rangle = \mathbb{K}_{i-1}[X_i]/\langle Q_{i,i-1}\rangle.$$

Applying $\Psi_{i,i}^{-1}$ gives us the image of A in $\mathbb{K}_i$, and applying Ψ_i finally brings it to $\mathbb{K}_0[X_i]/\langle Q_i\rangle$.

4.3 Towers from irreducible fibers

In this section we discuss another construction of the ℓ-adic tower based on work of Couveignes and Lercier [6]. The results of this section are summarized in rows 2, 4 and 5 of Table 4.1. This construction is not unrelated to the ones of the previous section, and indeed we will start by showing how those of Sections 4.2.2 and 4.2.3 reduce to it.
Here is the bottom line of Couveignes' and Lercier's idea. Let G, G' be integral algebraic $\mathbb{F}_q$-groups of the same dimension and let $\phi : G' \to G$ be a surjective, separable algebraic group morphism. Let ℓ be the degree of ϕ; then, the set of points $x \in G$ with fiber G'_x of cardinality ℓ is a nonempty open subset $U \subset G$. If the induced homomorphism $G'(\mathbb{F}_q) \to G(\mathbb{F}_q)$ of groups is not surjective then there are points of $G(\mathbb{F}_q)$ with fibers lying in algebraic extensions of $\mathbb{F}_q$. Assume that we are able to choose ϕ so that we can find one of these points contained in U, with an irreducible fiber, and apply a linear projection to this fiber (e.g., onto an axis). The resulting polynomial is irreducible of degree dividing ℓ (and expectedly equal to ℓ). If we can repeat the construction with a new map $\phi' : G'' \to G'$, and so on, the sequence of extensions makes an ℓ-adic tower over $\mathbb{F}_q$.

4.3.1 Towers from algebraic tori

In [6], Couveignes and Lercier explain how their idea yields the tower of Section 4.2.2. Consider the multiplicative group $\mathbb{G}_m$: this is an algebraic group of dimension one, and $\mathbb{G}_m(\mathbb{F}_q)$ has cardinality $q-1$. The ℓ-th power map defined by $\phi : X \mapsto X^\ell$ is a degree ℓ algebraic endomorphism of $\mathbb{G}_m$, surjective over the algebraic closure.
Suppose that ℓ divides $q-1$, and let η be a non ℓ-adic residue in $\mathbb{F}_q$ (η plays here the same role as y_0 in Section 4.2). For any $i > 0$, the fiber $\phi^{-i}(\eta)$ is defined by $X^{\ell^i} - \eta$: we recover the construction of Subsection 4.2.2.
More generally, following [26, 34], we let $k = \mathbb{F}_q$, $L = \mathbb{F}_{q^n}$ and $k \subset F \subsetneq L$. The Weil restriction $\mathrm{Res}_{L/k}\,\mathbb{G}_m$ is an *algebraic torus*, and the norm $\mathrm{N}_{L/F}$ induces a map $\mathrm{Res}_{L/k}\,\mathbb{G}_m \to \mathrm{Res}_{F/k}\,\mathbb{G}_m$. Define the *maximal torus* $\mathbb{T}_n$ as the intersection of the kernels of the maps $\mathrm{N}_{L/F}$ for all subfields F. Then $\mathbb{T}_n$ has dimension $\varphi(n)$, is isomorphic to $\mathbb{G}_m^{\varphi(n)}$ over the algebraic closure, and its k-rational points form a group of cardinality $\Phi_n(q)$:

$$\mathbb{T}_n(k) \cong \{\alpha \in L^* \mid \mathrm{N}_{L/F}(\alpha) = 1 \text{ for all } k \subset F \subsetneq L\}. \tag{4.7}$$

We now detail how the construction of Section 4.2.3 can be obtained by considering the torus $\mathbb{T}_2$; this will allow us to start completing the second row in Table 4.1.

Lemma 7. *Let $\Delta \in \mathbb{F}_q$ be a quadratic non-residue if $p \neq 2$, or such that* $\mathrm{tr}_{\mathbb{F}_q/\mathbb{F}_2}(\Delta) = 1$ *otherwise. Let $\delta = \sqrt{\Delta}$ or $\delta^2 + \delta = \Delta$ accordingly. The maximal torus $\mathbb{T}_2$ is isomorphic to the* Pell conic

$$C : \begin{cases} x^2 - \Delta y^2 = 4 & \text{if } p \neq 2, \\ x^2\Delta + xy + y^2 = 1 & \text{if } p = 2. \end{cases} \tag{4.8}$$

Multiplication in $\mathbb{T}_2$ induces a group law on C. The neutral element is $(2,0)$ if $p \neq 2$, or $(0,1)$ if $p=2$. The sum of two points $P=(x_1,y_1)$ and $Q=(x_2,y_2)$ is defined by

$$P \oplus Q = \begin{cases} \left(\dfrac{x_1x_2+\Delta y_1y_2}{2}, \dfrac{x_1y_2+x_2y_1}{2}\right) & \text{if } p \neq 2, \\ (x_1x_2+x_1y_2+x_2y_1,\ x_1x_2\Delta+y_1y_2) & \text{if } p=2. \end{cases}$$

Proof. The isomorphism follows by Weil restriction to $\mathbb{F}_q(\delta)/\mathbb{F}_q$ with respect to the basis $(1/2,\delta/2)$ if $p\neq 2$, or $(\delta,1)$ if $p=2$. Indeed, by virtue of Eq. (4.7), an element (x,y) of $\mathbb{F}_q(\delta)$ belongs to $\mathbb{T}_2$ if and only if its norm over $\mathbb{F}_q$ is 1. Let σ be the generator of $\mathrm{Gal}_{\mathbb{F}_q(\delta)/\mathbb{F}_q}$. For $p\neq 2$, clearly $\delta^\sigma=-\delta$. For $p=2$, by Artin-Schreier theory, $\mathrm{tr}_{\mathbb{F}_q(\delta)/\mathbb{F}_q}(\delta)=\mathrm{tr}_{\mathbb{F}_q/\mathbb{F}_2}(\Delta)=1$, hence $\delta^\sigma=1+\delta$. In both cases, Eq. (4.8) follows. The group law is obtained by direct calculation. □

Pell conics are a classic topic in number theory[20] and computer science, with applications to primality proving, factorization [19, 12] and cryptography [25].
As customary, we denote by $[n](x,y)$ the n-th scalar multiple of a point (x,y). $[n]$ is an endomorphism of C of degree n, separable if and only if $(n,p)=1$.

Lemma 8. *Let $P=(\alpha,\beta)$ be a point of C. The abscissa of $[n]P$ is given by $C_n(\alpha)$, where $C_n\in\mathbb{Z}[X]$ is the n-th Chebyshev polynomial, defined by $C_0=2$, $C_1=X$, and*

$$C_{n+1}=XC_n-C_{n-1}. \tag{4.9}$$

Proof. Induction on n. A detailed proof can be found in [30, Prop. 6.6]. □

Theorem 9. *Let $\eta\in\mathbb{F}_q(\delta)$ be a non ℓ-adic residue in $\mathbb{T}_2$, and let $P=(\alpha,\beta)$ be its image in $C/\mathbb{F}_q$. For any $i>0$, the polynomials $C_{\ell^i}-\alpha$ are irreducible. Their roots are the abscissas of the images in $C/\mathbb{F}_{q^{\ell^i}}$ of the ℓ^i-th roots of η.*

Proof. By [17, Th. VI.9.1], the polynomial $X^{\ell^i}-\eta$ is irreducible. Its roots correspond to the fiber $[\ell^i]^{-1}(P)$, and the Galois group of $\mathbb{F}_{q^{\ell^i}}/\mathbb{F}_q$ acts transitively on them.
Two points of C have the same abscissa if and only if they are opposite. But η is a non ℓ-adic residue, hence $\eta\neq\eta^{-1}$, and all the points in $[\ell^i]^{-1}(P)$ have distinct abscissa. By Lemma 8, $C_{\ell^i}-\alpha$ vanishes precisely on those abscissas and is thus irreducible. □

We can now apply our results to the computation of the polynomials Q_i and T_i of Section 4.2.3.

Corollary 10. *The polynomials $Q_{i,j}$ of Prop. 5 satisfy*

$$Q_{i,j}(X_i)=C_{\ell^{i-j}}(X_i)-x_j.$$

Proof. We have already shown that $\mathrm{N}_{\mathbb{K}_j/\mathbb{F}_q(x_j)}(y_j)=1$ for any j, thus y_j is a non ℓ-adic residue in $\mathbb{T}_2/\mathbb{F}_q(x_j)$. Independently of the characteristic and of the element $\Delta\in\mathbb{F}_q(x_j)$ chosen, the abscissa of the image of y_j in $C/\mathbb{F}_q(x_j)$ is $\mathrm{tr}_{\mathbb{K}_j/\mathbb{F}_q(x_j)}y_j=x_j$. The statement follows from the previous theorem. □

There is a folklore algorithm computing the n-th Chebyshev polynomial using $O(n)$ operations in $\mathbb{Z}$ [15]. We shall need a slightly better algorithm working modulo p.

Corollary 11. *The polynomials $Q_{i,j}$ can be computed using $O(\ell^{i-j})$ operations in $\mathbb{F}_p$.*

Proof. Let $C_n = \sum_i c_{n,i} X^{n-i}$. It is well known that $|c_{n+k,2k}|$ are the coefficients of the $(1,2)$-Pascal triangle, also called Lucas' triangle (see [30, Prop. 6.6] and [1]). It follows that

$$\frac{c_{n,2k+2}}{c_{n,2k}} = -\frac{(n-2k)(n-2k-1)}{(n-k-1)(k+1)},$$

which immediately gives the algorithm. Indeed, since we know the $c_{n,2k}$'s are the image mod p of integers, we compute them using multiplications and divisions in $\mathbb{Q}_p$ with relative precision 1. □

We are left with the problem of finding the non ℓ-adic residue η to initialize the tower. As before, this will be done by random sampling and testing.

Lemma 12. *Let $P = (\alpha, \beta)$ be a point on C. For any n, there is a formula to compute the abscissa of $[\pm n]P$, using $O(\log n)$ operations in $\mathbb{F}_q$, and not involving β.*

Proof. Observe that if $n = 2$, the abscissa of $[\pm 2]P$ is $\alpha^2 - 2$ (for any p). Let $P' = (\alpha', \beta')$, and let γ be the abscissa of $P \ominus P'$. By direct computation we find that the abscissa of $P \oplus P'$ is $\alpha\alpha' - \gamma$ (for any p); this formula is called a *differential addition*. Thus, $O(1)$ operations are needed for a doubling or a differential addition. To compute the abscissa of $[\pm n]P$, we use the ladder algorithm of [23], requiring $O(\log n)$ doublings and differential additions. □

Proposition 13. *The abscissa of a point $P \in C/\mathbb{F}_q$ satisfying the conditions of Theorem 9 can be found using $O_e(\log q)$ operations in $\mathbb{F}_q$.*

Proof. We randomly select $\alpha \in \mathbb{F}_q$ and test that it belongs to C. If $p \neq 2$, this amounts to testing that $\alpha^2 - 4$ is a quadratic non-residue in $\mathbb{F}_q$, a task that can be accomplished with $O(\log q)$ operations. If $p = 2$, by Artin-Schreier theory this is equivalent to $\mathrm{tr}_{\mathbb{F}_q/\mathbb{F}_2}(1/\alpha^2) = 1$, which can be tested in $O(\log q)$ operations in $\mathbb{F}_q$.
Then we check that P is a non ℓ-adic residue by verifying that $[(q+1)/\ell]P$ is not the group identity. By Lemma 12, this computation requires $O(\log q)$ operations. About half of the points of $\mathbb{F}_q$ are quadratic non-residues, and about $1 - 1/\ell$ of them are the abscissas of points with the required order, thus we expect to find the required element after $O_e(1)$ trials. □

It is natural to ask whether a similar construction could be applied to any ℓ. If r is the order of q modulo ℓ, the natural object to look at is $\mathbb{T}_r$, but here we are faced with two problems. First, multiplication by ℓ is now a degree $\ell^{\varphi(r)}$ map, thus its fibers have too many points; instead, isogenies of degree ℓ should be considered. Second, it is an open question whether $\mathbb{T}_r$ can be parameterized using $\varphi(r)$ coordinates; but even assuming it can be, we are still faced with the computation of a univariate annihilating polynomial for a set embedded in a $\varphi(r)$-dimensional space, a problem not known to be feasible in quasi-linear time. Studying this generalization is another natural follow-up to the present work.

$$\begin{tikzcd}
 & E_3 \arrow[dl, bend right, "\phi_3"'] & \\
E_4 \arrow[d, bend right, "\phi_4"'] & & E_2 \arrow[ul, bend right, "\phi_2"'] \\
E_0 \arrow[rr, bend right, "\phi_0"'] & & E_1 \arrow[u, bend right, "\phi_1"']
\end{tikzcd}$$

Figure 4.2: The isogeny cycle of E_0.

4.3.2 Towers from elliptic curves

Since it seems hard to deal with higher dimensional algebraic tori, it is interesting to look at other algebraic groups. Being one-dimensional, elliptic curves are good candidates. In this section, we quickly review Couveignes' and Lercier's construction, referring to [6] for details, and point out the modifications needed in order to build towers (as opposed to constructing irreducible polynomials).

Let ℓ be a prime different from p and not dividing $q-1$. Let E_0 be an elliptic curve whose cardinality over $\mathbb{F}_q$ is a multiple of ℓ. By Hasse's bound, this is only possible if $\ell \leq q+2\sqrt{q}+1$. An *isogeny* is an algebraic group morphism between two elliptic curves that is surjective in the algebraic closure. It is said to be rational over $\mathbb{F}_q$ if it is invariant under the q-th power map; such an isogeny exists if and only if the curves have the same number of points over $\mathbb{F}_q$. An isogeny of degree n is separable if and only if n is prime to p, in which case its kernel contains exactly n points. Because of the assumptions on ℓ, there exists an $e \geq 1$ such that, for any curve E isogenous to E_0, the $\mathbb{F}_q$-rational part of $E[\ell]$ is cyclic of order ℓ^e.

Suppose for simplicity, that $p \neq 2,3$ and let E_0 be expressed as the locus

$$E_0 : y^2 = x^3 + ax + b, \quad \text{with } a, b \in \mathbb{F}_q, \tag{4.10}$$

plus one point at infinity. We denote by H_0 the unique subgroup of $E_0/\mathbb{F}_q$ of order ℓ, and by ϕ_0 the unique isogeny whose kernel is H_0; we then label E_1 the image curve of ϕ_0. We go on denoting by H_i the unique subgroup of $E_i/\mathbb{F}_q$ of order ℓ, and by $\phi_i : E_i \to E_{i+1}$ the unique isogeny with kernel H_i. The construction is depicted in Figure 4.2.

Lemma 14. *Let $E_0, E_1, \ldots$ be defined as above, there exists $n \in O(\sqrt{q}\log(q))$ such that E_n is isomorphic to E_0.*

Proof. It is shown in [6, § 4] that the isogenies ϕ_i are *horizontal* in the sense of [16], hence they necessarily form a cycle. Let t be the trace of E_0, the length of the cycle is bounded by the class number of $\mathbb{Q}[X]/(X^2 - tX - q)$, thus by Minkowski's bound it is in $O(\sqrt{q}\log(q))$. □

In what follows, the index i is to be understood modulo the length of the cycle. This is a slight abuse, because E_n is isomorphic but not equal to E_0, but it does not hide any theoretical or computational difficulty.

Under the former assumptions, it is proved in [6, § 4] that if P is a point of E_i of order divisible by ℓ^e, if $\psi = \phi_{i-1} \circ \phi_{i-2} \circ \cdots \circ \phi_j$, then the fiber $\psi^{-1}(P)$ is irreducible and has cardinality ℓ^{i-j}. Knowing E_i, VÃl'lu's formulas [32] allow us to express the isogenies ϕ_i as rational fractions

$$\begin{aligned} \phi_i : E_i &\to E_{i+1}, \\ (x,y) &\mapsto \left(\frac{f_i(x)}{g_i(x)}, y\left(\frac{f_i(x)}{g_i(x)}\right)'\right), \end{aligned} \tag{4.11}$$

where g_i is the square polynomial of degree $\ell - 1$ vanishing on the abscissas of the affine points of H_i, and f_i is a polynomial of degree ℓ.

There is a subtle difference between our setting and Couveignes' and Lercier's. The goal of [6] is to compute an extension of degree ℓ^i of $\mathbb{F}_q$ for a fixed i: this can be done by going forward i times, then taking the fiber of a point of E_i by the isogenies $\phi_{i-1}, \dots, \phi_0$. In our case, we are interested in building extensions of degree ℓ^i *incrementally*, i.e. without any *a priori* bound on i. Thus, we have to walk *backwards* in the isogeny cycle: if $\eta \in \mathbb{F}_q$ is the abscissa of a point of E_0 of order $\ell^e \neq 2$, we will use the following polynomials to define the ℓ-adic tower:

$$\begin{aligned} T_1 &= f_{-1}(X_1) - \eta g_{-1}(X_1), \\ T_i &= f_{-i}(X_i) - X_{i-1} g_{-i}(X_i). \end{aligned}$$

The following theorem gives the time for building the tower; lift and push are detailed in the next section.

Theorem 15. *Suppose $4\ell \leq q^{1/4}$, and under the above assumption. Initializing the ℓ-adic tower requires $\tilde{O_e}(\ell \log^5(q) + \ell^3)$ bit operations; and building the i-th level requires $O_e(\ell^2 + \mathsf{M}(\ell)\log(\ell q) + \mathsf{M}(\ell^i)\log(\ell^i))$ operations in $\mathbb{F}_q$.*

Proof. For the initialization, [6, § 4.3] shows that if $4\ell \leq q^{1/4}$, a curve E_0 with the required number of points can be found in $\tilde{O_e}(\ell \log^5(q))$ bit operations. We also need to compute the ℓth modular polynomial Φ_ℓ mod p; for this, we compute it over $\mathbb{Z}$ with $\tilde{O}(\ell^3)$ bit operations [10], then reduce it modulo p.

To build the i-th level, we first need to find the equation of E_{-i}. For this, we evaluate Φ_ℓ at $j(E_{-i+1})$, using $O(\ell^2)$ operations. Lemma 14 implies that this polynomial has only two roots in $\mathbb{F}_q$, namely $j(E_{-i})$ and $j(E_{-i+2})$. We factor it using $O_e(\mathsf{M}(\ell)\log(\ell q))$ operations [33, Ch 14], and we take an arbitrary curve with j-invariant $j(E_{-i})$. Then we find an ℓ-torsion point using $O_e(\log q)$ operations, and apply VÃl'lu's formulas to compute ϕ_{-i}. We deduce the polynomial T_i, and Q_i is obtained using $O(\mathsf{M}(\ell^i)\log(\ell^i))$ operations using Algorithm 4 given in the next section. □

Remark. Instead of computing the cycle step by step, we could compute it entirely during the initialization phase, by using VÃl'lu's formulas alone to compute $E_1, E_2, \dots$ until we hit E_0 again. By doing so, we avoid using the modular polynomial Φ_ℓ at each new level. By Lemma 14, this requires $O_e(\ell\sqrt{q}\log(q))$ operations. This is not asymptotically good in q, but for practical values of q and ℓ the cycle is often small and this approach works well. This is accounted for in the last row of Table 4.1.

4.4 Lifting and pushing

The previous constructions of ℓ-adic towers based on irreducible fibers share a common structure that allows us to treat lifting and pushing in a unified way. Renaming the variables (X_{i-1}, X_i) as (X, Y), the polynomials (Q_{i-1}, Q_i, T_i) as (R, S, T), the extension at level i is described as

$$\mathbb{F}_q[Y]/\langle S(Y)\rangle \quad \text{and} \quad \mathbb{F}_q[X, Y]/\langle R(X), T(X, Y)\rangle,$$

with R of degree ℓ^{i-1}, S of degree ℓ^i, and where $T(X, Y)$ has the form $f(Y) - X g(Y)$, with $\deg(f) = \ell$, $\deg(g) < \ell$ and $\gcd(f, g) = 1$; possibly, $g = 1$. In all this section, f, g and their degree ℓ are fixed.
Lift is the conversion from the bivariate basis associated to the right-hand side to the univariate basis associated to the left-hand side; push is the inverse. Using the special shape of the polynomial T, they reduce to composition and decomposition of rational functions, as we show next. These results fill in all missing entries in the lift / push column of Table 4.1.

4.4.1 Lifting

Let P be in $\mathbb{F}_q[X, Y]$ and n be in $\mathbb{N}$, with $\deg(P, X) < n$. We define $P[f, g, n]$ as

$$P[f, g, n] = g^{n-1} P\left(\frac{f}{g}, Y\right) \in \mathbb{F}_q[X, Y].$$

If $P = \sum_{i=0}^{n-1} p_i(Y) X^i$, then $P[f, g, n] = \sum_{i=0}^{n-1} p_i f^i g^{n-1-i}$. We first give an algorithm to compute this expression, then show how to relate it to lifting; when $g = 1$, Algorithm 4 reduces to a well known algorithm for polynomial composition [33, Ex. 9.20].

Algorithm 4 Compose

```
Input: $P \in \mathbb{F}_q[X, Y]$, $f, g \in \mathbb{F}_q[Y]$, $n \in \mathbb{N}$
if $n = 1$ then
    return $P$
else
    $m \leftarrow \lceil n/2 \rceil$
    Let $P_0, P_1$ be such that $P = P_0 + X^m P_1$
    $Q_0 \leftarrow$ Compose($P_0, f, g, m$)
    $Q_1 \leftarrow$ Compose($P_1, f, g, n - m$)
    $Q \leftarrow Q_0 g^{n-m} + Q_1 f^m$
    return $Q$
end if
```

Theorem 16. *On input P, f, g, n, with $\deg(P, X) < n$ and $\deg(P, Y) < \ell$, Algorithm 4 computes $Q = P[f, g, n]$ using $O(\mathsf{M}(\ell n) \log(n))$ operations in $\mathbb{F}_q$.*

Proof. If $n = 1$, the theorem is obvious. Suppose $n > 1$, then P_0 and P_1 have degrees less than m and $n - m$ respectively. By induction hypothesis,

$$Q_0 = P_0[f, g, m] = \sum_{i=0}^{m-1} p_i f^i g^{m-1-i},$$
$$Q_1 = P_1[f, g, n-m] = \sum_{i=0}^{n-m-1} p_{i+m} f^i g^{n-m-1-i}.$$

Hence,

$$Q = \sum_{i=0}^{m-1} p_i f^i g^{n-1-i} + \sum_{i=0}^{n-m-1} p_{i+m} f^{i+m} g^{n-m-1-i} = P[f, g, n].$$

The only step that requires a computation is Step 8, costing $O(\mathsf{M}(\ell n))$ operations in $\mathbb{F}_q$. The recursion has depth $\log(n)$, hence the overall complexity is $O(\mathsf{M}(\ell n)\log(n))$. □

Corollary 17. *At level i, one can perform the lift operation using $O(\mathsf{M}(\ell^i)\log(\ell^i))$ operations in $\mathbb{F}_q$.*

Proof. We start from an element α written on the bivariate basis, that is, represented as $A(X, Y)$ with $\deg(A, X) < n = \ell^{i-1}$ and $\deg(A, Y) < \ell$ (note that $\ell n = \ell^i$). We compute the univariate polynomials $A^\star = A[f, g, n]$ and $\gamma = g^{n-1}$ using $O(\mathsf{M}(\ell^i)\log(\ell^i))$ operations in $\mathbb{F}_q$; then the lift of α is $A^\star/\gamma$ modulo S. The inverse of γ is computed using $O(\mathsf{M}(\ell n)\log(\ell n))$ operations, and the multiplication adds an extra $O(\mathsf{M}(\ell n))$. □

4.4.2 Pushing

We first deal with the inverse of the question dealt with in Theorem 16: starting from $Q \in \mathbb{F}_q[Y]$, reconstruct $P \in \mathbb{F}_q[X, Y]$ such that $Q = P[f, g, n]$. When $g = 1$, Algorithm 5 reduces to Algorithm 9.14 of [33].

Algorithm 5 Decompose

```
Input: $Q, f, g, h \in \mathbb{F}_q[Y]$, $n \in \mathbb{N}$
 1. if $n = 1$ then
 2.     return $Q$
 3. else
 4.     $m \leftarrow \lceil n/2 \rceil$
 5.     $u \leftarrow 1/g^{n-m} \bmod f^m$
 6.     $Q_0 \leftarrow Qu \bmod f^m$
 7.     $Q_1 \leftarrow (Q - Q_0 g^{n-m}) \operatorname{div} f^m$
 8.     $P_0 \leftarrow$ Decompose($Q_0, f, g, h, m$)
 9.     $P_1 \leftarrow$ Decompose($Q_1, f, g, h, n-m$)
10.     return $P_0 + X^m P_1$
11. end if
```

Theorem 18. *On input Q, f, g, h, n, with $\deg(Q) < \ell n$ and $h = 1/g \bmod f$, Algorithm 5 computes a polynomial $P \in \mathbb{F}_q[X, Y]$ such that $\deg(P, X) < n$, $\deg(P, Y) < \ell$ and $Q = P[f, g, n]$ using $O(\mathsf{M}(\ell n)\log(n))$ operations in $\mathbb{F}_q$.*

Proof. We prove the theorem by induction. If $n = 1$, the statement is obvious, so let $n > 1$. The polynomials Q_0 and Q_1 verify $Q = Q_0 g^{n-m} + Q_1 f^m$. By construction, Q_0 has degree less than ℓm. Since $\deg(g) < \ell$, this implies that $Q_0 g^{n-m}$ has degree less than ℓn; thus, Q_1 has degree less than $\ell(n-m)$. By induction, P_0 and P_1 have degree less than m, resp. $n-m$, in X, and less than ℓ in Y, and

$$\begin{aligned} Q_0 &= P_0[f, g, m] = \sum_{i=0}^{m-1} p_{0,i} f^i g^{m-1-i}, \\ Q_1 &= P_1[f, g, n-m] = \sum_{i=0}^{n-m-1} p_{1,i} f^i g^{n-m-1-i}. \end{aligned}$$

Hence, $P = P_0 + X^m P_1$ has degree less than n in X and less than ℓ in Y, and the following proves correctness:

$$\begin{aligned} P[f, g, n] &= \sum_{i=0}^{m-1} p_{0,i} f^i g^{n-1-i} + \sum_{i=m}^{n-1} p_{1,i-m} f^i g^{n-1-i} \\ &= P_0[f, g, m] g^{n-m} + P_1[f, g, n-m] f^m \\ &= Q. \end{aligned}$$

At Step 5, we do as follows: starting from $h = 1/g \bmod f$, we deduce $1/g^{n-m} \bmod f$ in time $O(\mathsf{M}(\ell)\log(n))$ by binary powering mod f. We also compute g^{n-m} in time $O(\mathsf{M}(\ell n))$ by binary powering, and we use Newton iteration (starting from $1/g^{n-m} \bmod f$) to deduce $1/g^{n-m} \bmod f^m$ in time $O(\mathsf{M}(\ell n))$. All other steps cost $O(\mathsf{M}(\ell n))$; the recursion has depth $\log(n)$, so the total cost is $O(\mathsf{M}(\ell n)\log(n))$. □

Corollary 19. *At level i, one can perform the push operation using $O(\mathsf{M}(\ell^i)\log(\ell^i))$ operations in $\mathbb{F}_q$.*

Proof. Given α represented by a univariate polynomial $A(Y)$ of degree less than ℓn, with $n = \ell^{i-1}$. We compute g^{n-1} and $A^\star = g^{n-1} A \bmod S$ using $O(\mathsf{M}(\ell^i))$ operations. Then, we compute $h = 1/g \bmod f$ in time $O(\mathsf{M}(\ell)\log(\ell))$ and apply Algorithm 5 to $A^\star$, f, g, h and n. The result is a bivariate polynomial B, representing α on the bivariate basis. The dominant phase is Algorithm 5, costing $O(\mathsf{M}(\ell^i)\log(\ell^i))$ operations in $\mathbb{F}_q$. □

4.5 Implementation

To demonstrate the interest of our constructions, we made a very basic implementation of the towers of Sections 4.3.1 and 4.3.2 in Sage [31]. It relies on Sage's default implementation of quotient rings of $\mathbb{F}_p[X]$, which itself uses NTL [29] for $p = 2$ and FLINT [13] for other primes. Towers based on elliptic curves are constructed using the algorithm described in Remark 4.3.2. The source code is available at `http://defeo.github.io/towers`

We compare our implementation to three ways of constructing ℓ-adic towers in Magma. First, one may construct the levels from bottom to top using the finite field constructor `GF()`. For the parameters we used, Magma uses tables of precomputed Conway polynomials and automatically computes embeddings on creation, see `http://magma.maths.usyd.edu.au/magma/releasenotes/2/14`. The second approach constructs the highest level of the tower first, then all the lower levels using the `sub<>` constructor. The last one constructs the levels from bottom to top using random dense polynomials and calls the `Embed()` function; we do not count the time for finding the irreducible polynomials.

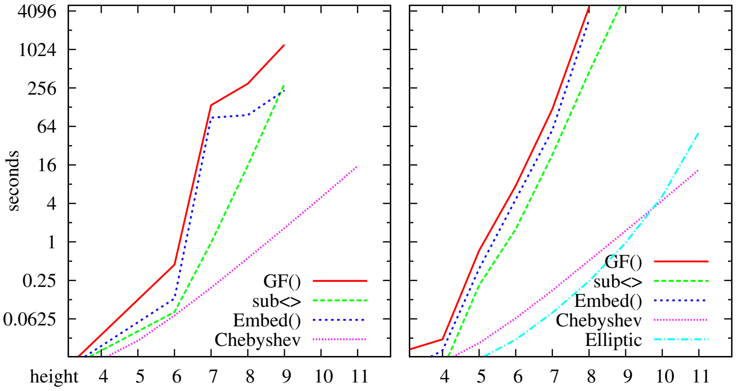


Figure 4.3: Times for building 3-adic towers on top of $\mathbb{F}_2$ (left) and $\mathbb{F}_5$ (right), in Magma (first three lines) and using our code.

We ran tests on an Intel Xeon E5620 clocked at 2.4 GHz, using Sage 5.5 and Magma 2.18.12. The time required for the creation of 3-adic towers of increasing height is summarized in Figure 4.3; the timings of our algorithms are labeled Chebyshev and Elliptic. Computations that took more than 4GB RAM were interrupted.

Despite its simplicity, our code consistently outperforms Magma on creation time. On the other hand, lift and push operations take essentially no time in Magma, while in all the tests of Figure 4.3 we measured a running time almost perfectly linear for one push followed by one lift, taking approximately $70\mu s$ per coefficient (this is in the order of a second around level 10). Nevertheless, the large gain in creation time makes the difference in lift and push tiny, and we are convinced that an optimized C implementation of the algorithms of Section 4.4 would match Magma's performances.

Acknowledgments. We acknowledge support from NSERC, the CRC program, and ANR through the ECLIPSES project under Contract ANR-09-VERS-018. De Feo would like to thank Antoine Joux and Jérôme Plût for fruitful discussions. We are grateful to the reviewers for their remarks.

Bibliography

[1] Arthur T. Benjamin. The Lucas triangle recounted. In *Congressus Numerantium*, volume 200, pages 169–177, 2010.

[2] Wieb Bosma, John Cannon, and Catherine Playoust. The MAGMA algebra system I: the user language. *J. Symbolic Comput.*, 24(3-4):235–265, 1997.

[3] Wieb Bosma, John Cannon, and Allan Steel. Lattices of compatibly embedded finite fields. *J. Symbolic Comput.*, 24(3-4):351–369, 1997.

[4] David G. Cantor. On arithmetical algorithms over finite fields. *J. Combin. Theory Ser. A*, 50(2):285–300, 1989.

[5] Jean-Marc Couveignes. Isomorphisms between Artin-Schreier towers. *Math. Comp.*, 69(232):1625–1631, 2000.

[6] Jean-Marc Couveignes and Reynald Lercier. Fast construction of irreducible polynomials over finite fields. *To appear in the Israel Journal of Mathematics*, July 2011.

[7] Luca De Feo. Fast algorithms for computing isogenies between ordinary elliptic curves in small characteristic. *Journal of Number Theory*, 131(5):873–893, May 2011.

[8] Luca De Feo and Éric Schost. Fast arithmetics in Artin-Schreier towers over finite fields. *J. Symbolic Comput.*, 47(7):771–792, 2012.

[9] J. Doliskani and É. Schost. A note on computations in degree 2^k-extensions of finite fields, 2012. Manuscript.

[10] Andreas Enge. Computing modular polynomials in quasi-linear time. *Math. Comp.*, 78(267):1809–1824, 2009.

[11] P. Gaudry and É. Schost. Point-counting in genus 2 over prime fields. *J. Symbolic Comput.*, 47(4):368âĂŞ400, 2012.

[12] Samuel A. Hambleton. Generalized Lucas-Lehmer tests using Pell conics. *Proceedings of the American Mathematical Society*, 140:2653–2661, 2012.

[13] William Hart. Fast library for number theory: an introduction. *Mathematical Software–ICMS 2010*, pages 88–91, 2010.

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

[15] Wolfram Koepf. Efficient computation of chebyshev polynomials in computer algebra. *Computer Algebra Systems: A Practical Guide.*, pages 79–99, 1999.

[16] David Kohel. *Endomorphism rings of elliptic curves over finite fields.* PhD thesis, University of California at Berkley, 1996.

[17] Serge Lang. *Algebra.* Springer, 3rd edition, January 2002.

[18] R. Lebreton and É. Schost. Algorithms for the universal decomposition algebra. In *ISSAC'12*, pages 234–241. ACM, 2012.

[19] Franz Lemmermeyer. Conics - a Poor Man's Elliptic Curves, 2003.

[20] Hendrick W. Lenstra. Solving the Pell equation. *Notices of the AMS*, 49(2):182–192, 2002.

[21] Hendrick W. Lenstra and Bart De Smit. Standard models for finite fields: the definition, 2008.

[22] T. Lickteig and M.-F. Roy. SylvesterâĂŞhabicht sequences and fast cauchy index computation. *J. Symbolic Comput.*, 31(3):315 – 341, 2001.

[23] Peter L. Montgomery. Speeding the pollard and elliptic curve methods of factorization. *Math. Comp.*, 48(177), 1987.

[24] D. Reischert. Asymptotically fast computation of subresultants. In *ISSAC*, pages 233–240. ACM, 1997.

[25] Karl Rubin and Alice Silverberg. Torus-Based cryptography. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 349–365, Berlin, Heidelberg, 2003. Springer Berlin / Heidelberg.

[26] Karl Rubin and Alice Silverberg. Algebraic tori in cryptography. In *In High Primes and Misdemeanours: Lectures in Honour of the 60th birthday of Hugh Cowie Williams*, volume 41 of *Fields Institute Communications*. AMS, 2004.

[27] Victor Shoup. New algorithms for finding irreducible polynomials over finite fields. *Math. Comp.*, 54:435–447, 1990.

[28] Victor Shoup. Fast construction of irreducible polynomials over finite fields. *J. Symbolic Comput.*, 17(5):371–391, 1994.

[29] Victor Shoup. NTL: A library for doing number theory. `http://www.shoup.net/ntl`, 2003.

[30] Joseph H Silverman. *The arithmetic of dynamical systems*, volume 241 of *Graduate Texts in Mathematics*. Springer, 2007.

[31] William A. Stein and Others. *Sage Mathematics Software (Version 5.5)*. The Sage Development Team, 2013.

[32] Jean Vélu. Isogénies entre courbes elliptiques. *Comptes Rendus de l'Académie des Sciences de Paris*, 273:238–241, 1971.

[33] Joachim von zur Gathen and Jürgen Gerhard. *Modern computer algebra*. Cambridge University Press, New York, NY, USA, 1999.

[34] Valentin E. Voskresenskiĭ. *Algebraic groups and their birational invariants*, volume 179. American Mathematical Society, 1998.

Chapter 5

Fast arithmetic for the algebraic closure of finite fields

5.1 Introduction

Several computer algebra systems or libraries, such as Magma [3], Sage [39], NTL [37], PARI [31] or Flint [24], offer built-in features to build and compute in arbitrary finite fields. At the core of these designs, one finds algorithms for building irreducible polynomials and algorithms to compute embeddings and isomorphisms. The system used in Magma (one of the most complete we know of) is described in [4].

Previous algorithms typically rely on linear algebra techniques, for instance to describe embeddings or isomorphisms (this is the case for the algorithms in [4], but also for those in [28, 1]). Unfortunately, linear algebra techniques have cost at least quadratic in the degree of the extensions we consider, and (usually) quadratic memory requirements. Our goal here is to replace linear algebra by polynomial arithmetic, exploiting fast polynomial multiplication to obtain algorithms of quasi-linear complexity. As we will see, we meet this goal for several, but not all, operations.

Setup. Let p be a prime (that will be fixed throughout this paper). We are interested in describing extensions $\mathbb{F}_{p^n}$ of $\mathbb{F}_p$; such an extension has dimension n over $\mathbb{F}_p$, so representing an element in it involves n base field elements.

It is customary to use polynomial arithmetic to describe these extensions (but not necessary: Lenstra's algorithm [28] uses a multiplication tensor). For an extension degree n, a first step is to construct an irreducible polynomial Q_n of degree n in $\mathbb{F}_p[x]$. Identifying $\mathbb{F}_{p^n}$ with $\mathbb{F}_p[x]/\langle Q_n\rangle$, operations $(+, \times, \div)$ in $\mathbb{F}_p[x]/\langle Q_n\rangle$ all take quasi-linear time in n.

However, this is not sufficient: we also want mechanisms for e.g. field embeddings. Given irreducible polynomials Q_m and Q_n over $\mathbb{F}_p$, with $\deg(Q_m) = m$ dividing $\deg(Q_n) = n$, there exist algorithms to embed $\mathbb{F}_p[x]/\langle Q_m\rangle$ in $\mathbb{F}_p[x]/\langle Q_n\rangle$ (for the system to be consistent, these embeddings must be *compatible* [4]). However, most algorithms use linear algebra techniques.

To bypass these issues, we use an approach inspired by Shoup's algorithm for computing irreducible polynomials [35, 36] (see also [16, 29]): first reduce to the case of prime power degrees, then use composita techniques, in a manner that ensures compatibility of the embeddings automatically.

Background: towers. Suppose that for any prime ℓ, an *ℓ-adic tower* over $\mathbb{F}_p$ is available. By this, we mean a family of polynomials $(T_{\ell,i})_{i\geq 1}$, with $T_{\ell,i} \in \mathbb{F}_p[x_1,\ldots,x_i]$, monic of degree ℓ in x_i, such that for all i the ideal $\langle T_{\ell,1},\ldots,T_{\ell,i}\rangle$ is maximal in $\mathbb{F}_p[x_1,\ldots,x_i]$. Our model of the field with p^{ℓ^i} elements could then be $\mathbb{K}_{\ell^i} = \mathbb{F}_p[x_1,\ldots,x_i]/\langle T_{\ell,1},\ldots,T_{\ell,i}\rangle$. Arithmetics in this type of representation can be performed using the triangular-set techniques, see [30], but we prefer to work with univariate polynomials (the cost of arithmetic operations is generally higher in the multivariate basis).
For $1 \leq i \leq n$, let then $Q_{\ell,i}$ be the minimal polynomial of x_i in the extension $\mathbb{K}_{\ell^n}/\mathbb{F}_p$. This polynomial does not depend on n, but only on i; it is monic, irreducible of degree ℓ^i in $\mathbb{F}_p[x_i]$ and allows us to define $\mathbb{F}_{p^{\ell^i}}$ as $\mathbb{F}_p[x_i]/\langle Q_{\ell,i}\rangle$. For $1 \leq i \leq j \leq n$, let further $Q_{\ell,i,j-i}$ be the minimal polynomial of x_j in the extension $\mathbb{F}_p[x_i]/\langle Q_{\ell,i}\rangle \hookrightarrow \mathbb{K}_{\ell^n}$ (as above, it does not depend on n). This polynomial is monic, irreducible of degree ℓ^{j-i} in $\mathbb{F}_{p^{\ell^i}}[x_j] = \mathbb{F}_p[x_i]/\langle Q_{\ell,i}\rangle[x_j]$.
Thus, $\mathbb{F}_p[x_j]/\langle Q_{\ell,j}\rangle$ and $\mathbb{F}_p[x_i,x_j]/\langle Q_{\ell,i}, Q_{\ell,i,j-i}\rangle$ are two models for $\mathbb{F}_{p^{\ell^j}}$. Provided conversion algorithms between these representations are available, we can perform embeddings (that will necessarily be compatible) between different levels of the ℓ-adic tower, i.e. extensions of degrees $(\ell^i)_{i\geq 1}$.
Such towers, together with efficient conversion algorithms, were constructed in the cases $\ell = p$ in [13, 15, 19], $\ell = 2$ in [21], and for other values of ℓ in [18]. Thus, it remains to give algorithms to "glue" towers defined for different values of ℓ. This is the purpose of this paper.

Our contribution. The algorithms used to construct towers are inspired by those used in [35, 36, 16] to build irreducible polynomials. Also used in these references is the following idea: let $Q_m(x)$ and $Q_n(y)$ be irreducible polynomials over $\mathbb{F}_p$, with coprime degrees $m, n > 1$, and having respectively $(a_i)_{1\leq i\leq m}$ and $(b_j)_{1\leq j\leq n}$ as roots in an algebraic closure of $\mathbb{F}_p$. Then their *composed product* $Q_{mn} = \prod_{1\leq i\leq m, 1\leq j\leq n}(z - a_i b_j)$ is irreducible of degree mn in $\mathbb{F}_p[z]$.
In this paper, we use an *algebraic complexity model*, where the cost of an algorithm is counted in terms of the number of operations $(+, \times, \div)$ in $\mathbb{F}_p$. If the goal is building irreducible polynomials, then computing Q_{mn} is enough: an algorithm given in [6] has quasi-linear cost in mn. Our goal here is to give algorithms for further operations: computing embeddings of the form $\varphi_x : \mathbb{F}_p[x]/\langle Q_m\rangle \to \mathbb{F}_p[z]/\langle Q_{mn}\rangle$ or $\varphi_y : \mathbb{F}_p[y]/\langle Q_n\rangle \to \mathbb{F}_p[z]/\langle Q_{mn}\rangle$, and the isomorphism $\Phi : \mathbb{F}_p[x,y]/\langle Q_m, Q_n\rangle \to \mathbb{F}_p[z]/\langle Q_{mn}\rangle$ or its inverse.
Standard solutions to these questions exist, using *modular composition* techniques: once the image $S = \Phi(x)$ is known, computing $\varphi_x(a)$ amounts to computing $a(S) \bmod Q_{mn}$; similarly, computing $\Phi(b)$, for b in $\mathbb{F}_p[x,y]/\langle Q_m, Q_n\rangle$, amounts to computing $b(S,T) \bmod Q_{mn}$, with $T = \Phi(y)$. This can be done using the Brent and Kung algorithm [11]: the resulting cost is $O(mn^{(\omega+1)/2}) \subset O(mn^{1.69})$ for φ_x (see the analysis in [36]) and $O((mn)^{(\omega+1)/2}) \subset O(m^{1.69}n^{1.69})$ for Φ or its inverse [33]. Here, we denote by ω a constant in $(2,3]$ such that one can multiply matrices of size m over any ring A using $O(m^\omega)$ operations $(+,\times)$ in A; using the algorithms of [14, 40], we can take $\omega \leq 2.38$.

Our main result improves on these former ones. We denote by $\mathsf{M} : \mathbb{N} \to \mathbb{N}$ a function such that for any ring A, polynomials in $A[x]$ of degree at most n can be multiplied in $\mathsf{M}(n)$ operations $(+,\times)$ in A, and we make the usual super-linearity assumptions on M [22, Chapter 8].

Theorem 20. *One can apply φ_x (resp. φ_y) to an element of $\mathbb{F}_p[x]/\langle Q_m\rangle$ (resp. $\mathbb{F}_p[x]/\langle Q_n\rangle$), or invert it on its image, using $O(n\mathsf{M}(m)+m\mathsf{M}(n))$ operations in $\mathbb{F}_p$.*
Suppose that $m \le n$. Then one can apply Φ to an element of $\mathbb{F}_p[x,y]/\langle Q_m, Q_n\rangle$ or invert it using either $O(m^2\mathsf{M}(n))$ or $O(\mathsf{M}(mn)n^{1/2}+\mathsf{M}(m)n^{(\omega+1)/2})$ operations in $\mathbb{F}_p$.

Using the $O\tilde{}$ notation to neglect polylogarithmic factors, we can take $\mathsf{M}(n) \in O\tilde{}(n)$. Our algorithm for embeddings and their inverses has quasi-linear cost $O\tilde{}(mn)$. Those for Φ or Φ^{-1} have respective costs $O\tilde{}(m^2n)$ and $O\tilde{}(mn^{(\omega+1)/2})$; the minimum of the two is in $O\tilde{}((mn)^{2\omega/(\omega+1)})$; for $\omega \in (2,3]$, the resulting exponent is in $(1.333\ldots, 1.5]$.
If $S=\Phi(x)$ and $T=\Phi(y)$ are known, a result by Kedlaya and Umans [25] for modular composition, and its extension in [32], yield an algorithm with *bit complexity* essentially linear in mn and $\log(p)$ on a RAM. Unfortunately, making these algorithms competitive in practice is challenging; we are not aware of any implementation of them. It is also worth noting that our algorithms apply in a more general setting than finite fields (mild assumptions are required).

Outline. Section 5.2 presents basic algorithms for polynomials and their transposes. Section 5.3 introduces the main idea behind our algorithms: the trace induces a duality on algebras of the form $\mathbb{F}_p[x]/\langle Q\rangle$, and some conversion algorithms are straightforward in dual bases; the algorithms are detailed in Section 5.4. Section 5.5 explains how the results in this paper can be used in order to construct the algebraic closure of $\mathbb{F}_p$. We conclude with experimental results.

5.2 Preliminaries

We recall first previous results concerning polynomial arithmetic and transposition of algorithms. In all this section, a ground field k, not necessarily finite, is fixed. For integers m,n, we denote by $k[x]_m$ (resp. $k[x,y]_{m,n}$) the set of polynomials P in $k[x]$ with $\deg(P) < m$ (resp. P in $k[x,y]$ with $\deg(P,x) < m$ and $\deg(P,y) < n$).

5.2.1 Polynomial multiplication and remainder

We start with some classical algorithms and their complexity. For all the algorithms that follow, all polynomials are written on the canonical monomial basis (this is innocuous for the moment, but other bases will be discussed below).
The product of two polynomials of respective degrees at most m and n can be computed in $\mathsf{M}(\max(m,n))$ operations in k. If P is a monic polynomial of degree m in $k[x]$, for $n \ge 1$, we let $\mathsf{rem}(.,P,n)$ be the operator

$$\begin{array}{rccc} \mathsf{rem}(.,P,n): & k[x]_n & \to & k[x]_m \\ & a & \mapsto & a \bmod P. \end{array}$$

For $n \le m$, this is free of cost. For $n > m$, this can be computed in time $O(n\mathsf{M}(m)/m)$ using the Cook-Sieveking-Kung algorithm and blocking techniques [5, Ch. 5.1.3]. Defining $A = k[x]/\langle P\rangle$, and choosing a fixed $b \in A$, we can then define the mapping $\mathsf{mulmod}(., b, P)$, which maps $a \in A$ to $ab \bmod P$; it can be computed in time $O(\mathsf{M}(m))$. Finally, given an integer m, the reversal operator in length m is

$$\begin{array}{rccc} \mathsf{rev}(., m): & k[x]_m & \to & k[x]_m \\ & a & \mapsto & x^{m-1}a(1/x). \end{array}$$

5.2.2 Duality and the transposition principle

The *transposition principle* is an algorithmic result which states that, given an algorithm that performs a matrix-vector product $u \mapsto Mu$, one can deduce an algorithm with essentially the same cost which performs the transposed matrix-vector product $v \mapsto M^t v$ [12, Ch. 13].

Following [20], we give here a more abstract presentation of the transposition principle, using the algebraic theory of duality (see [9, Ch. IX.1.8]). The added level of abstraction will pay off by greatly simplifying the proofs of the next sections.

Let E and F be k-vector spaces, with $\dim(E) = \dim(F) < \infty$, and suppose that $\langle ., . \rangle : E \times F \to k$ is a non-degenerate bilinear form. Then, to any vector space basis $\xi = (\xi_i)_i$ of E, we can associate a unique *dual basis* $\xi^* = (\xi_i^*)_i$ of F such that $\langle \xi_i, \xi_j^* \rangle = \delta_{i,j}$ (the Kronecker symbol). In other words, given a in F, the coefficients (a_i) of a on the basis ξ^* are given by $a_i = \langle \xi_i, a \rangle$.

For example, denote by E^* the dual space of E, i.e. the k-linear forms on E. The bilinear form on $E \times E^*$ defined by with $\langle v, \ell \rangle = \ell(v)$ for all $v \in E$ and $\ell \in E^*$ is non-degenerate. This is indeed the canonical example, and any non-degenerate form, is isomorphic to this one. We will see in the next section another family of examples, with $E = F$.

Let E', F' be two further vector spaces, with $\dim(E') = \dim(F') < \infty$ and let $\langle ., . \rangle'$ be a bilinear form $E' \times F' \to k$. Then, to any linear mapping $u : E \to E'$, one associates its *dual* (with respect to $\langle ., . \rangle$ and $\langle ., . \rangle'$), which is a linear mapping $u^t : F' \to F$ characterized by the equality $\langle u(a), b' \rangle' = \langle a, u^t(b') \rangle$, for all $a \in E$ and $b' \in F'$.

Let as above ξ be a basis of E, and let ξ^* be the dual basis of F; consider as well a basis υ of E' and its dual basis υ^* of F'. If M is the matrix of u in the bases (ξ, υ), the matrix of u^t in the bases (υ^*, ξ^*) is the transpose of M.

As presented in [8, 20], the transposition principle is an algorithmic technique that, given an algorithm to compute $u : E \to E'$ in the bases (ξ, υ), yields an algorithm for the dual map $u^t : F' \to F$ in the bases (υ^*, ξ^*). The two algorithms have same cost, up to $O(\dim(E) + \dim(E'))$. In a nutshell, starting from an algorithm relying on a few basic operations (such as polynomial or matrix multiplication), its transpose is obtained by transposing each basic subroutine, then reversing their order.

Let us briefly review the transposes of operations described in the previous subsection. The transpose of polynomial multiplication is described in [8]; it is closely related to the *middle product* [23]. Let next P be monic of degree m, and define $A = k[x]/\langle P \rangle$. As shown in [8], the dual map of rem

$$\mathsf{rem}^t(., P, n) : \ A^* \ \to \ k[x]_n^*$$

is equivalent to *linear sequence extension*: it takes as input the initial m values of a linear recurring sequence of minimal polynomial P, and outputs its first n values. The transposed version of the Cook-Sieveking-Kung fast Euclidean division algorithm yields an algorithm with cost $O(n\mathsf{M}(m)/m)$ operations in k [41, 38].

For a fixed $b \in A$, the transpose of mulmod is the map

$$\begin{array}{rccc} \mathsf{mulmod}^t(.,b,P): & A^* & \to & A^* \\ & \ell & \mapsto & b\cdot\ell, \end{array}$$

where $b\cdot\ell$ is defined by $(b\cdot\ell)(a)=\ell(ab)$. Algorithms for mulmod^t have been subject to much research (for instance, Berlekamp's *bit serial multiplication* [2] is a popular arithmetic circuit for mulmod^t in the case $k=\mathbb{F}_2$); algorithms of cost $O(\mathsf{M}(m))$ are given in [38, 8].
Lastly, the reversal operator on $k[x]_m$ is its own transpose.

5.3 Trace and duality

Next, we discuss some classical facts about the trace form, and give algorithms to change between monomial bases and their duals. In all this section, k is a perfect field. General references for the following are [26, 17].

Traces in reduced algebras. Let s be a positive integer and I a zero dimensional radical ideal in $k[x_1,\dots,x_s]$. Thus, $A=k[x_1,\dots,x_s]/I$ is a reduced k-algebra of finite dimension d, where d is the cardinality of $V=V(I)\subset \overline{k}^s$ (in general, A is not a field).
Let a be in A. As we did in the case of one variable, we associate to a the endomorphism of multiplication-by-a $M_a : A\to A$ given by $M_a(b)=ab$. Even though A may not be a field, we still define the *minimal polynomial* of a as the minimal polynomial of M_a; since I is radical, this polynomial is squarefree, with roots $a(x)$, for x in V. Similarly, the *trace* of a is the trace of M_a, and denote it by $\tau_I(a)$. Because I is radical, the trace defines a non-degenerate bilinear form on $A\times A$, given by $\langle a,b\rangle_I=\tau_I(ab)$.
Thus, to any basis $\xi=(\xi_i)_{0\le i<d}$ of A, one can associate a dual basis $\xi^*=(\xi_i^*)_{0\le i<d}$, such that $\langle \xi_i,\xi_j^*\rangle_I=\delta_{i,j}$ for all i,j. It will be useful to keep in mind that for $a\in A$, its expression on the dual basis ξ^* is $a=\sum_{0\le i<d}\langle a,\xi_i\rangle_I\xi_i^*$.
We now describe algorithms for converting between the monomial basis and its dual, in two particular cases, involving respectively univariate and bivariate polynomials. In both cases, our conclusion will be that such conversions have quasi-linear complexity.

Univariate conversion. Let P be monic of degree m and squarefree in $k[x]$, and define $A=k[x]/\langle P\rangle$. We denote by P' its derivative and by τ_P the trace modulo the ideal $\langle P\rangle$.
The k-algebra A is endowed with the canonical monomial basis $\xi=(x^i)_{0\le i<m}$. In view of what was said in the previous subsection, the coefficients of an element $a\in A$ on the dual basis ξ^* are the traces $\tau_P(ax^i)_{0\le i<m}$. The following lemma shows that the generating series of these traces is rational, with a known denominator; this will be the key to the conversion algorithm. This is a restatement of well-known results, see for instance the proof of [34, Theorem 3.1].

Lemma 21. *For a in A, the following holds in $k[[x]]$:*

$$\sum_{i\ge 0}\tau_P(ax^i)x^i=\frac{\mathsf{rev}(P'a \bmod P, m)}{\mathsf{rev}(P,m+1)}.$$

Some well-known algorithms to convert between ξ and ξ^* follow easily. In these algorithms, and all that follows, input and output are vectors (written in sans serif font).

Algorithm 6 MonomialToDual(a, P)

```
Input: $\mathsf{a} = (a_i)_{0 \le i < m} \in k^m$, $P$ monic squarefree in $k[x]$ of degree $m$
Output: $(\tau_P(a x^i))_{0 \le i < m}$, with $a = \sum_{0 \le i < m} a_i x^i$
  $T = 1/\mathsf{rev}(P, m+1) \bmod x^m$
  $b = \mathsf{rev}(P' \sum_{0 \le i < m} a_i x^i \bmod P, m)\, T \bmod x^m$
  return $(\mathsf{coefficient}(b, x^i))_{0 \le i < m}$
```

Algorithm 7 DualToMonomial(b, P)

```
Input: $\mathsf{b} = (b_i)_{0 \le i < m} \in k^m$, $P$ monic squarefree in $k[x]$ of degree $m$
Output: $(a_i)_{0 \le i < m}$ such that $\tau_P(\sum_{0 \le i < m} a_i x^{i+j}) = b_j$ for all $j$
  $S = 1/P' \bmod P$
  $b = \mathsf{rev}(P, m+1) \sum_{0 \le i < m} b_i x^i \bmod x^m$
  $c = \mathsf{rev}(b, m)$
  $d = c\, S \bmod P$
  return $(\mathsf{coefficient}(d, x^i))_{0 \le i < m}$
```

Lemma 22. *Algorithms 6 and 7 are correct. The former uses $O(\mathsf{M}(m))$ operations in k, the latter $O(\mathsf{M}(m)\log(m))$. If the polynomial $S = 1/P' \bmod P$ is known, the running time of Algorithm 7 drops to $O(\mathsf{M}(m))$.*

Proof. Correctness follows from Lemma 21. Once we point out that power series inversion modulo x^m can be done in time $O(\mathsf{M}(m))$, the running time analysis of the former is straightforward. For Algorithm 7, the dominant part is the computation of S, which takes time $O(\mathsf{M}(m)\log(m))$ by fast XGCD; all other steps take $O(\mathsf{M}(m))$ operations in k. □

Bivariate conversions. Now we consider two monic squarefree polynomials P in $k[x]$ of degree m, and Q in $k[y]$ of degree n. We define $A = k[x,y]/I$, with $I = \langle P, Q \rangle$, then A has the canonical monomial basis $(x^i y^j)_{0 \le i < m, 0 \le j < n}$. We denote by τ_I the trace modulo I, and by τ_P and τ_Q the traces modulo respectively $\langle P \rangle$ and $\langle Q \rangle$.

In addition to its monomial basis, A can be endowed with a total of four natural bases, which are described as follows. Let $\xi = (x^i)_{0 \le i < m}$ and $\upsilon = (y^i)_{0 \le j < n}$ be the monomial bases of respectively $k[x]/\langle P \rangle$ and $k[y]/\langle Q \rangle$; let ξ^* and υ^* be their respective dual bases, with respect to τ_P and τ_Q. The monomial basis seen above is $\xi \otimes \upsilon$; the other combinations $\xi^* \otimes \upsilon$, $\xi \otimes \upsilon^*$ and $\xi^* \otimes \upsilon^*$ are bases of A as well. After a precomputation of cost $O(\mathsf{M}(m)\log(m) + \mathsf{M}(n)\log(n))$, Lemma 22 shows that conversions between any pair of these bases can be done using $O(n\mathsf{M}(m) + m\mathsf{M}(n))$ operations in k (by applying the univariate conversion algorithms n times x-wise and / or m times y-wise). Using fast multiplication, this is quasi-linear in the dimension mn of A.

The following easy lemma will help us exhibit the duality relationships between these bases; it follows from the fact that A is the tensor product of $k[x]/\langle P \rangle$ and $k[y]/\langle Q \rangle$.

Lemma 23. *Let b be in $k[x]/\langle P \rangle$ and c in $k[y]/\langle Q \rangle$. Then we have $\tau_I(bc) = \tau_P(b)\, \tau_Q(c)$.*

This lemma implies that $\xi \otimes \upsilon$ and $\xi^* \otimes \upsilon^*$ are dual to one another with respect to $\langle .,. \rangle_I$, as are $\xi^* \otimes \upsilon$ and $\xi \otimes \upsilon^*$.

5.4 Embedding and isomorphism

This section contains the main algorithms of this paper. We consider two squarefree polynomials $P(x)$ and $Q(y)$ of respective degrees m and n, with coefficients in a perfect field k. Let us then set $A = k[x,y]/I$, where I is the ideal $\langle P(x), Q(y)\rangle$ in $k[x,y]$. In all this section, *we assume that xy is a generator of A as a k-algebra.*

The main example we have in mind is the following: k is a finite field and both P and Q are irreducible, with $\gcd(m,n) = 1$. Then our assumption is satisfied and in addition A is a field, namely, the *compositum* of the fields $k[x]/\langle P\rangle$ and $k[y]/\langle Q\rangle$, see [10]. More generally, if we let $(r_i)_{i<m}$ be the roots of P in an algebraic closure of k, and let $(s_j)_{j<n}$ be the roots of Q, then as soon as the products $r_i s_j$ are pairwise distinct, xy generates A as a k-algebra.

Let $R \in k[z]$ be the minimal polynomial of xy in the extension A/k (equivalently, the roots of R are the products $r_i s_j$); this polynomial is known as the *composed product* of P and Q, and we will denote it $R = P \odot Q$. As k-algebras, we have $A \simeq k[x]/\langle R\rangle$, so there exist embeddings φ_x, φ_y, and an isomorphism Φ of the form

$$\begin{array}{rrcl} \varphi_x : & k[x]/\langle P\rangle & \to & k[z]/\langle R\rangle, \\ \varphi_y : & k[y]/\langle Q\rangle & \to & k[z]/\langle R\rangle, \\ \Phi : & A = k[x,y]/\langle P, Q\rangle & \to & k[z]/\langle R\rangle \\ & xy & \longleftrightarrow & z. \end{array}$$

In this section, we give algorithms for computing R, applying φ_x, φ_y and their sections, and finally Φ and its inverse. Except from the computation of R, these are all linear algebra problems. If R and the images $S = \Phi(x), T = \Phi(y)$ are known, then as was explained in the introduction, direct solutions are available for both φ_x (or φ_y) and Φ – modular composition – but none of these approaches have a quasi-linear running time.

We take a different path. Our algorithms have quasi-linear running time for φ_x and φ_y and improve on the Brent-Kung algorithm for Φ. Put together, Lemmas 25 to 29 below prove Theorem 20. One of the key aspects of these algorithms is that some are written in the usual monomial bases, whereas others are naturally expressed in the corresponding dual bases. From the complexity point of view, this is not an issue, since we saw that all change-of-bases can be done in quasi-linear time.

In what follows, we write $\tau_P, \tau_Q, \tau_R, \tau_I$ for the traces modulo the ideals $\langle P\rangle \subset k[x]$, $\langle Q\rangle \subset k[y]$, $\langle R\rangle \subset k[z]$ and $I = \langle P, Q\rangle \subset k[x,y]$; the corresponding bilinear forms are denoted by $\langle .,.\rangle_P, \ldots$

We let $\xi = (x^i)_{0\le i<m}$, $\upsilon = (y^i)_{0\le j<n}$ and $\zeta = (z^i)_{0\le i<mn}$ be the monomial bases of respectively $k[x]/\langle P\rangle$, $k[y]/\langle Q\rangle$ and $k[z]/\langle R\rangle$. We also let $\xi^* = (\xi_i^*)_{0\le i<m}$, $\upsilon^* = (\upsilon_i^*)_{0\le i<n}$ and $\zeta^* = (\zeta_i^*)_{0\le i<mn}$ be the dual bases, with respect to respectively $\langle .,.\rangle_P$, $\langle .,.\rangle_Q$ and $\langle .,.\rangle_R$.

Finally, we denote by $\mathsf{u}_P \in k^m$ the vector of the coordinates of $1 \in k[x]/\langle P\rangle$ on the dual basis ξ^*; the vector u_Q is defined similarly. These vectors can both be computed in quasi-linear time, since we have, for instance, $\mathsf{u}_P = \mathsf{MonomialToDual}((1,0,\ldots,0),P)$. Thus, in what follows, we assume that these vectors are known.

5.4.1 Embedding and computing R

We first show how to compute the embeddings φ_x and φ_y, and their inverses in quasi-linear time in mn. We actually give a slightly more general algorithm, which computes the restriction of Φ to the set

$$\Pi = \{bc \mid b \in k[x]/\langle P\rangle,\ c \in k[y]/\langle Q\rangle\} \subset k[x,y]/\langle P,Q\rangle.$$

We will use the following lemma, which results from the base independence of the trace (the second equality is Lemma 23).

Lemma 24. *Let b be in $k[x]/\langle P\rangle$ and c in $k[y]/\langle Q\rangle$. Then we have $\tau_R(\Phi(bc)) = \tau_I(bc) = \tau_P(b)\,\tau_Q(c)$.*

An easy consequence is that $\tau_R(z^i) = \tau_P(x^i)\tau_Q(y^i)$. From this lemma, we also immediately deduce Algorithm 8, which computes the image in $k[z]/\langle R\rangle$ of any element of Π, with inputs and outputs written on dual bases.

```
Algorithm 8 Embed(b, c, r)
Input: $\mathsf{b} = (b_i)_{0\le i<m} \in k^m$, $\mathsf{c} = (c_i)_{0\le i<n} \in k^n$ an optional integer $r \ge mn$ set to $r = mn$ by default
Output: $\mathsf{a} = (a_i)_{0\le i<r} \in k^r$
  $(t_i)_{0\le i<r} = \mathsf{rem}^t(\mathsf{b}, P, r)$
  $(u_i)_{0\le i<r} = \mathsf{rem}^t(\mathsf{c}, Q, r)$
  return $(t_i u_i)_{0\le i<r}$
```

Lemma 25. *Let $b \in k[x]/\langle P\rangle$ and $c \in k[y]/\langle Q\rangle$. Given the coefficients* b *and* c *of respectively b and c in the bases ξ^* and υ^*,* Embed(b, c, r) *computes $a_i = \tau_R\big(\Phi(bc)z^i\big)$ for $0 \le i < r$ in time $O(r(\mathsf{M}(m)/m + \mathsf{M}(n)/n))$. If $r = mn$, $(a_i)_{0\le i<mn}$ are the coefficients of $\Phi(bc)$ in the basis ζ^*.*

Proof. Recall that for $0 \le i < m$, $b_i = \tau_P(bx^i)$, and that for $0 \le i < n$, $c_i = \tau_Q(cy^i)$. By definition of rem^t, the sequences (t_i) and (u_i) encode the same traces, but up to index r. By Lemma 24, the algorithm correctly computes

$$\big(\tau_P(bx^i)\tau_Q(cy^i)\big)_{i<r} = \big(\tau_R(\Phi(bc)z^i)\big)_{i<r}.$$

For $r = mn$, this is indeed the representation of $\Phi(bc)$ on the dual basis ζ^* of $k[z]/\langle R\rangle$. The cost of the calls to rem^t is in Section 5.2.2; the last step takes r multiplications in k. □

In particular, the map φ_x is computed as Embed($\cdot$, u_Q), and the map φ_y as Embed(u_P, $\cdot$). Another interesting consequence is that, when A is known to be a field, Embed allows us to compute R, using the Berlekamp-Massey algorithm.

```
Algorithm 9 ComputeR(P, Q)
Input: $P$ in $k[x]$, $Q$ in $k[y]$
Output: $R$ in $k[z]$
  $(t_i)_{0\le i<2mn} = \mathsf{Embed}(\mathsf{u}_P, \mathsf{u}_Q, 2mn)$,
  return BerlekampMassey($(t_i)_{0\le i<2mn}$)
```

Indeed, in this case, $\mathsf{Embed}(\mathsf{u}_P, \mathsf{u}_Q, 2mn)$ computes the sequence $(\tau_R(z^i))_{0\le i<2mn}$. If we know that A is a field, R is irreducible, so the minimal polynomial of this sequence (which is computed by the Berlekamp-Massey algorithm) is precisely R. A fast variant of Berlekamp-Massey algorithm gives the running time of $O(\mathsf{M}(mn)\log(mn))$ operations in k. This algorithm for computing R is well-known; see for instance [6] for a variant using power series exponentials instead of Berlekamp-Massey's algorithm (that applies in large enough characteristic) and [7] for the specific case of finite fields of small characteristic.

For the inverse of say φ_x, we take a in $k[z]/\langle R\rangle$ of the form $a=\varphi_x(b)$, and compute b. Using the equality of Lemma 24 in the form $\tau_P(bx^i)=\tau_R(az^i)/\tau_Q(y^i)$ would lead to a simple algorithm, but some traces $\tau_Q(y^i)$ may vanish.

We take a different path. Let c be a fixed element in $k[y]/\langle Q\rangle$ such that $\tau_Q(c)=1$; we will take for c the first element υ_0^* of the dual basis of $k[y]/\langle Q\rangle$, but this is not necessary. Let us denote by $\epsilon : k[x]/\langle P\rangle \to k[z]/\langle R\rangle$ the mapping defined by $\epsilon(b)=\Phi(bc)$, and let $\epsilon^t : k[z]/\langle R\rangle \to k[x]/\langle P\rangle$ be its dual map with respect to the bilinear forms $\langle .,.\rangle_P$ and $\langle .,.\rangle_R$. Then, for b and b' in $k[x]/\langle P\rangle$, we have

$$\langle b,b'\rangle_P=\tau_P(bb')=\tau_P(bb')\tau_Q(c)=\tau_R(\Phi(bb'c))=\langle \epsilon(b),\Phi(b')\rangle_R=\langle b,\epsilon^t(\Phi(b'))\rangle_P,$$

where the third equality comes from Lemma 24. Using the non-degeneracy of $\langle .,.\rangle_P$, we get $\epsilon^t(\Phi(b'))=b'$, that is, $\epsilon^t(\varphi_x(b'))=b'$. Thus, ϵ^t is an inverse of φ_x on its image.

Writing $\mathsf{c}=(1,0,\ldots,0)$, we remark that $\mathsf{Embed}(.,\mathsf{c})$ precisely computes the mapping $b\mapsto\epsilon(b)$. Since Embed is written in the dual bases, the discussion of Section 5.2.2 shows that transposing this algorithm (with respect to b) yields an algorithm for ϵ^t written in the monomial bases.

Algorithm 10 Project(a)

```
Input: $\mathsf{a}=(a_i)_{0\le i<mn}\in k^{mn}$
Output: $\mathsf{b}=(b_i)_{0\le i<m}\in k^m$
1. $\mathsf{c}=(1,0,\ldots,0)$
2. $(u_i)_{0\le i<mn}=\mathsf{rem}^t(\mathsf{c},Q,mn)$
3. $d=\sum_{i=0}^{mn-1}a_iu_ix^i \bmod P$
4. return $(\mathsf{coefficient}(d,i))_{0\le i<m}$
```

Lemma 26. *Let $b\in k[x]/\langle P\rangle$ and $a=\varphi_x(b)$. Given the coefficients* a *of a in the basis $\zeta=(z^i)_{0\le i<mn}$,* Project(a) *computes the coefficients of b in the basis $\xi=(x^i)_{0\le i<m}$ using $O(n\mathsf{M}(m)+n\mathsf{M}(n))$ operations in k.*

Proof. We show correctness using transposition techniques as in [8]. For fixed c, $\mathsf{Embed}(\mathsf{b},\mathsf{c})$ is linear in b and can be written as $\pi_{\mathsf{c}}\circ\mathsf{rem}^t$, where π_{c} is the map that multiplies a vector in k^{mn} coefficient-wise by $(\tau_Q(cy^i))_{i<mn}$, for $c=\sum_{0\le i<n}c_i\upsilon_i^*$; hence, its transpose is $\mathsf{rem}\circ\pi_{\mathsf{c}}^t$. It is evident that $\pi_{\mathsf{c}}^t=\pi_{\mathsf{c}}$ (since π_{c} is a diagonal map), whereas rem is just reduction modulo P. These correspond to steps 3 and 4. The discussion above now proves that the output is $\epsilon^t(a)$. The cost analysis is similar to the one in Lemma 25. □

5.4.2 Isomorphism

We are not able to give an algorithm for Φ that would be as efficient as those for embedding; instead, we provide two algorithms, with different domains of applicability. In what follows, without loss of generality, *we assume that* $m \leq n$.

Recall that $\xi \otimes \upsilon$, $\xi^* \otimes \upsilon$, $\xi \otimes \upsilon^*$ and $\xi^* \otimes \upsilon^*$ are four bases of A, with $(\xi \otimes \upsilon, \xi^* \otimes \upsilon^*)$ and $(\xi^* \otimes \upsilon, \xi \otimes \upsilon^*)$ being two pairs of dual bases with respect to $\langle .,. \rangle_I$. Our algorithms will exploit all these bases; this is harmless, since conversions between these bases have quasi-linear complexity.

Before giving the details of the algorithms, we make an observation similar to the one we did regarding the transpose of Embed. Let Φ^t be the dual map of Φ with respect to $\langle .,. \rangle_I$ and $\langle .,. \rangle_R$. Then, for any $b, b' \in k[z]/\langle R \rangle$, we have:

$$\langle b, b' \rangle_I = \tau_I(bb') = \tau_R(\Phi(bb')) = \langle \Phi(b), \Phi(b') \rangle_R = \langle b, \Phi^t(\Phi(b')) \rangle_I;$$

hence, $\Phi^t = \Phi^{-1}$. If $\mathbf{b}$ and $\mathbf{b}^*$ are two bases of $A = k[x,y]/I$, dual with respect to $\langle .,. \rangle_I$ (such as the ones seen above) and if $\mathbf{c}$ and $\mathbf{c}^*$ are two bases of $k[z]/\langle R \rangle$, dual with respect to $\langle .,. \rangle_R$, the previous equality, together with the transposition principle, shows the following: if we have an algorithm for Φ, expressed in the bases $(\mathbf{b}, \mathbf{c})$, transposing it yields an algorithm for Φ^{-1}, expressed in the bases $(\mathbf{c}^*, \mathbf{b}^*)$.

First case: m is small. We start by a direct application of the results in the previous subsection, which is well-suited to situations where m is small compared to n.

Let b be in $k[x,y]/I$ and let $a = \Phi(b)$. Writing $b = \sum_{0 \leq i < m} b_i x^i$, with all b_i in $k[y]/\langle Q \rangle$, we obtain a straightforward algorithm to compute a: compute all $\Phi(b_i x^i)$ using Embed, then sum. Since Embed takes its inputs written on the dual bases, the algorithm requires that all b_i be written on the dual basis of $k[y]/\langle Q \rangle$ (equivalently, the input is given on the basis $\xi \otimes \upsilon^*$ of A). We also use the fact that the expression of x^i on the dual basis ξ^* is u_P shifted by i positions to give a more compact algorithm, called Phi1.

Transposing this algorithm then gives an algorithm for Φ^{-1}. Its input is given on the monomial basis $(z^i)_{0 \leq i < mn}$ of $k[z]/\langle R \rangle$; the output is written on the basis $\xi^* \otimes \upsilon$ of A.

Algorithm 11 Phi1(b)

```
Input: $\mathsf{b} = (b_{i,j})_{0 \leq i < m, 0 \leq j < n} \in k^{m \times n}$
Output: $\mathsf{a} = (a_i)_{0 \leq i < mn} \in k^{mn}$
$(u_i)_{0 \leq i < m(n+1)-1} = \mathsf{rem}^t(\mathsf{u}_P, P, m(n+1)-1)$
$(a_i)_{0 \leq i < mn} = (0, \dots, 0)$
for $0 \leq i < m$ do
    $(t_j)_{0 \leq j < mn} = \mathsf{rem}^t((b_{i,j})_{0 \leq j < n}, Q, mn)$
    $(a_j)_{0 \leq j < mn} = (a_j + t_j u_{i+j})_{0 \leq j < mn}$
end for
return $(a_i)_{0 \leq i < mn}$
```

Algorithm 12 InversePhi1(a)

```
Input: $\mathsf{a} = (a_i)_{0 \leq i < mn} \in k^{mn}$
Output: $\mathsf{b} = (b_{i,j})_{0 \leq i < m, 0 \leq j < n} \in k^{m \times n}$
```

```
$(u_i)_{0\le i<m(n+1)-1} = \mathsf{rem}^t(\mathsf{u}_P, P, m(n+1)-1)$
for $i = m-1,\dots,0$ do
    $d = \sum_{0\le j<mn} a_j u_{i+j} y^j \bmod Q$
    $(b_{i,j})_{0\le j<n} = (\mathsf{coefficient}(d,j))_{0\le j<n}$
end for
return $(b_{i,j})_{0\le i<m, 0\le j<n}$
```

Lemma 27. *Let $b \in k[x,y]/I$. Given the coefficients* b *of b in the basis $\xi \otimes \upsilon^*$,* $\mathsf{Phi1}(\mathsf{b})$ *computes the coefficients of $\Phi(b)$ in the basis ζ^* using $O(m^2\mathsf{M}(n))$ operations in k.*
Let $a \in k[z]/\langle R\rangle$. Given the coefficients a *of a in the basis $\zeta = (z^i)_{0\le i<mn}$,* $\mathsf{InversePhi1}(\mathsf{a})$ *computes the coefficients of $\Phi^{-1}(a)$ in the basis $\xi \otimes \upsilon^*$ using $O(m^2\mathsf{M}(n))$ operations in k.*

Proof. Correctness of $\mathsf{Phi1}$ follows from the previous discussion; the most expensive step is m calls to rem^t, for a cumulated cost of $O(m^2\mathsf{M}(n))$.
The correctness of the transposed algorithm is proved as in Lemma 26, observing that it consists of the line-by-line transposition of $\mathsf{Phi1}$. The running time analysis is straightforward: the dominant cost is that of m remainders, each of which costs $O(m\mathsf{M}(n))$. □

Second case: m is not small. The previous algorithms are most efficient when m is small; now, we propose an alternative solution that does better when m and n are of the same order of magnitude (with still $m \le n$).
This approach is based on baby steps / giant steps techniques, as in Brent and Kung's modular composition algorithm, but uses the fact that $z = \Phi(xy)$ to reduce the cost. Given b in $A = k[x,y]/\langle P,Q\rangle$, let us write

$$\begin{aligned} b &= \sum_{i=0}^{m-1}\sum_{j=0}^{n-1} b_{i,j} x^i y^j \\ &= \sum_{i=0}^{m-1}\sum_{j=0}^{n-1} b_{i,j} x^i y^i y^{j-i} \\ &= \sum_{h=-m+1}^{n-1}\sum_{i=0}^{m-1} b_{i,i+h} (xy)^i y^h \\ &= \frac{1}{y^{m-1}} \sum_{h=0}^{m+n-2} c_h(xy) y^h, \end{aligned}$$

with $c_h(z) = \sum_{0\le i<m} b_{i,i+h-m+1} z^i$ for all h (undefined indices are set to zero). Hence $a = \Phi(b)$ has the form

$$a = \frac{1}{T^{m-1}}\widetilde{a} \mod R \quad \text{with} \quad \widetilde{a} = \sum_{h=0}^{m+n-2} c_h T^h,$$

where $T = \Phi(y)$. We use baby steps / giant steps techniques from [27] (inspired by Brent and Kung's algorithm) to compute a, reducing the problem to polynomial matrix multiplication. Let $n' = m+n-1$, $p = \lceil \sqrt{n'}\rceil$ and $q = \lceil n'/p\rceil$, so that $n \le n' \le 2n-1$ and $p \simeq q \simeq \sqrt{n}$. For baby steps, we compute the polynomials $T_i = T^i \bmod R$, which have degree at most $mn-1$; we

write $T_i = \sum_{0 \le j < n} T'_{i,j} z^{jm}$, with $T'_{i,j}$ of degree less than m, and build the polynomial matrix $M_{T'}$ with entries $T'_{i,j}$. We define the matrix $M_C = [c_{iq+j}]_{0 \le i < p, 0 \le j < q}$ containing the polynomials c_h organized in a row-major fashion, and compute the product $M_V = M_C M_T$. We can then construct polynomials from the rows of M_V, and conclude with giant steps using Horner's scheme.
The previous discussion leads to Algorithm 13. Remark that input *and* output are written on the monomial bases.

Algorithm 13 Phi2(b)

```
Input: $\mathsf{b} = (b_{i,j})_{0 \le i < m, 0 \le j < n} \in k^{m \times n}$
Output: $\mathsf{a} = (a_i)_{0 \le i < mn} \in k^{mn}$
$n' = m + n - 1$, $p = \lceil \sqrt{n'} \rceil$, $q = \lceil n'/p \rceil$
$\mathsf{y} = \mathsf{MonomialToDual}((0, 1, 0, \dots, 0), Q)$
$T = \mathsf{DualToMonomial}(\mathsf{Embed}(\mathsf{u}_p, \mathsf{y}), R)$
$U = 1/T \bmod R$
$T' = [T^i \bmod R]_{0 \le i \le q}$
$M_{T'} = [T'_{i,j}]_{0 \le i < q, 0 \le j < n}$ {$T'_{i,j}$ are defined in the text}
$M_C = [c_{iq+j}]_{0 \le i < p, 0 \le j < q}$ {$c_h$ are defined in the text}
$M_V = M_C M_{T'}$
$V = [\sum_{0 \le j < n} M_{V\,i,j} z^{jm}]_{0 \le i < p}$
$V' = [V_i \bmod R]_{0 \le i < p}$
$a = 0$
for $i = p - 1, \dots, 0$ do
    $a = T'_q a + V'_i \bmod R$
end for
$a = a\, U^{m-1} \bmod R$
return $(\mathsf{coefficient}(a, i))_{0 \le i < mn}$
```

Algorithm 14 InversePhi2(a)

```
Input: $\mathsf{a} = (a_i)_{0 \le i < mn} \in k^{mn}$
Output: $\mathsf{b} = (b_{i,j})_{0 \le i < m, 0 \le j < n} \in k^{m \times n}$
$n' = m + n - 1$, $p = \lceil \sqrt{n'} \rceil$, $q = \lceil n'/p \rceil$
$\mathsf{y} = \mathsf{MonomialToDual}((0, 1, 0, \dots, 0), Q)$
$T = \mathsf{DualToMonomial}(\mathsf{Embed}(\mathsf{u}_p, \mathsf{y}), R)$
$U = 1/T \bmod R$
$T' = [T^i \bmod R]_{0 \le i \le q}$
$M_{T'} = [T'_{i,j}]_{0 \le i < q, 0 \le j < n}$ {$T'_{i,j}$ as defined above}
$\mathsf{a} = \mathsf{mulmod}^t(\mathsf{a}, U^{m-1}, R)$
for $i = 0, \dots, p - 1$ do
    $V'_i = \mathsf{a}$
    $\mathsf{a} = \mathsf{mulmod}^t(\mathsf{a}, T'_q, R)$
end for
$V = [\mathsf{rem}^t(V'_i, R, mn + m - 1)]_{0 \le i < p}$
$M_V = [(V_i)_{jm, \dots, jm+2m-2}]_{0 \le i < p, 0 \le j < n}$
```

```
14. $M_C = \mathsf{mul}^t(M_V, M_{T'}, m-1, m)$
15. $c = [M_{C\,0,0}, \dots, M_{C\,0,q-1}, \dots, M_{C\,p-1,q-1}]$
16. return $[\mathsf{coefficient}(c_{i-j+m-1}, i)]_{0\le i<m, 0\le j<n}$
```

Lemma 28. *Let $b \in k[x,y]/I$. Given the coefficients* b *of b in the basis $\xi \otimes \upsilon = (x^i y^j)_{0\le i<m, 0\le j<n}$,* Phi2(b) *computes the coefficients of $\Phi(b)$ in the basis $\zeta = (z^i)_{0\le i<mn}$ in $O(\mathsf{M}(mn)n^{1/2}+\mathsf{M}(m)n^{(\omega+1)/2})$ operations in k.*

Proof. Correctness follows from the discussion prior to the algorithm. As to the cost analysis, remark first that $n' = O(n)$, and that p and q are both $O(\sqrt{n})$. Steps 4 and 15 cost $O(\mathsf{M}(mn)\log(mn))$ operations. Steps 5 (the baby steps) and the loop at Step 12 (the giant steps) cost $O(\sqrt{n}\mathsf{M}(mn))$. The dominant cost is the matrix product at Step 8, which involves matrices of size $O(\sqrt{n}) \times O(\sqrt{n})$ and $O(\sqrt{n}) \times O(n)$, with polynomial entries of degree m: using block matrix multiplication in size $O(\sqrt{n})$, this takes $O(\mathsf{M}(m)n^{(\omega+1)/2})$ operations in k. □

As before, writing the transpose of this algorithm gives us an algorithm for Φ^{-1}, this time written in the dual bases. The process is the same for the previous transposed algorithms we saw, involving line-by-line transposition. The only point that deserves mention is Step 14, where we transpose polynomial matrix multiplication; it becomes a similar matrix product, but this time involving transposed polynomial multiplications (with degree parameters $m-1$ and m). The cost then remains the same, and leads to Lemma 29.

Lemma 29. *Let $a \in k[z]/\langle R\rangle$. Given the coefficients* a *of a in the basis ζ^*,* InversePhi2(a) *computes the coefficients of $\Phi^{-1}(a)$ in the basis $\xi^* \otimes \upsilon^*$ in $O(\mathsf{M}(mn)n^{1/2}+\mathsf{M}(m)n^{(\omega+1)/2})$ operations in k.*

5.5 The algebraic closure of $\mathbb{F}_p$

In this section, we explain how the algorithms of Section 5.4 can be used in order to construct and work in arbitrary extensions of $\mathbb{F}_p$, when used in conjunction with algorithms for *ℓ-adic towers* over $\mathbb{F}_p$. Space constraints prevent us from giving detailed algorithms, so we only outline the construction. We reuse definitions given in the introduction relative to ℓ-adic towers: polynomials $T_{\ell,i}$, $Q_{\ell,i}$ and $Q_{\ell,i,j-i}$ and fields $\mathbb{K}_{\ell^i} = \mathbb{F}_p[x_1,\dots,x_i]/\langle T_{\ell,1},\dots,T_{\ell,i}\rangle$. We also assume that algorithms for embeddings or change of basis in ℓ-adic towers are available (as in [18] and references therein).

Setup. For ℓ prime and $i \ge 1$, the residue class of x_i in $\mathbb{K}_{\ell^i}$ will be written x_{ℓ^i}. For a positive integer $m = \ell_1^{e_1}\cdots\ell_r^{e_r}$, with ℓ_i pairwise distinct primes and e_i positive integers, $\mathbb{K}_m$ denotes the tensor product $\mathbb{K}_{\ell_1^{e_1}} \otimes \cdots \otimes \mathbb{K}_{\ell_r^{e_r}}$; this is a field with p^m elements. If m divides n, then $\mathbb{K}_m$ embeds in $\mathbb{K}_n$. Taking the direct limit of all $\mathbb{K}_m$ under such embeddings, we get an algebraic closure $\mathbb{K}$ of $\mathbb{F}_p$. The residue classes written x_{ℓ^e} in $\mathbb{K}_{\ell^e}$ all lie in $\mathbb{K}$ and are still written x_{ℓ^e}.
For any integer m of the form $m = \ell_1^{e_1}\cdots\ell_r^{e_r}$ with ℓ_i's pairwise distinct primes, we write $x_m = x_{\ell_1^{e_1}}\cdots x_{\ell_r^{e_r}} \in \mathbb{K}$.

Minimal polynomials. We discuss first minimal polynomials of monomials in $\mathbb{K}$ over $\mathbb{F}_p$. Take x_{ℓ^e} in $\mathbb{K}$, with ℓ prime. By construction, its minimal polynomial over $\mathbb{F}_p$ is $Q_{\ell,e}$, irreducible of degree ℓ^e in (say) $\mathbb{F}_p[z]$. Next, consider a term x_m, with $m = \ell_1^{e_1} \cdots \ell_r^{e_r}$, with ℓ_i's pairwise distinct primes. It equals $x_{\ell_1^{e_1}} \cdots x_{\ell_r^{e_r}}$, so it is a root of the composed product $Q_m = Q_{\ell_1,e_1} \odot \cdots \odot Q_{\ell_r,e_r}$. In Section 5.4, we pointed out that Q_m is irreducible of degree $m = \ell_1^{e_1} \cdots \ell_r^{e_r}$ in $\mathbb{F}_p[z]$, so it must be the minimal polynomial of x_m over $\mathbb{F}_p$. In particular, this implies that $\mathbb{F}_p(x_m)$ is a field with p^m elements, and that if we consider terms x_m and x_n, with m dividing n, then x_m is in $\mathbb{F}_p(x_n)$.

Note that this process of constructing irreducible polynomials over $\mathbb{F}_p$ is already in [35, 36, 16].

Embedding and change of basis. Consider a sequence $e = (e_1, \ldots, e_t)$ of positive integers, and let $n = e_1 \cdots e_t$. The set

$$B_e = \{x_{e_1}^{a_1} x_{e_1e_2}^{a_2} \cdots x_{e_1\cdots e_t}^{a_t} \mid 0 \le a_i < e_i \text{ for all } i\}$$

is a basis of $\mathbb{F}_p(x_n)$. Important examples are sequences of the form $e = (e_1)$, with thus $n = e_1$, for which B_e is the univariate basis $(x_n^i)_{0 \le i < n}$. Also useful for us are sequences $e = (e_1, e_2)$; letting $m = e_1$ and $n = e_1 e_2$, B_e is the bivariate basis $(x_m^i x_n^j)_{0 \le i < m, 0 \le j < n/m}$.

Consider sequences $d = (d_1, \ldots, d_s)$ and $e = (e_1, \ldots, e_t)$, with $m = d_1 \cdots d_s$ and $n = e_1 \cdots e_t$, and suppose that m divides n. The linear mapping $\mathbb{F}_p^m \to \mathbb{F}_p^n$ that describes the embedding $\mathbb{F}_{p^m} \to \mathbb{F}_{p^n}$ in the bases B_d and B_e is denoted by $\Phi_{e,d}$; when $m = n$, it is an isomorphism, with inverse $\Phi_{d,e}$. More generally, as soon as this expression makes sense, we have $\Phi_{f,d} = \Phi_{f,e} \circ \Phi_{e,d}$, so these mappings are compatible.

To conclude this section, we describe how the algorithms of this paper can be used in this framework to realize some particular cases of mappings $\Phi_{d,e}$ (more general examples can be deduced readily).

Embedding. Consider two integers m, n with m dividing n. We describe here how to embed $\mathbb{F}_p(x_m)$ in $\mathbb{F}_p(x_n)$, that is, how to compute $\Phi_{(n),(m)}$. Without loss of generality, we may assume that $n = m\ell$, with ℓ prime.

Assume first that $\gcd(m, \ell) = 1$. Since then $x_n = x_m x_\ell$, and we have access to the polynomials Q_m, Q_ℓ and Q_n (see above), we just apply the embedding algorithm of Section 5.4.

Suppose now that ℓ divides m, so $m = m'\ell^k$, with m', ℓ coprime. Using one of the inverse isomorphism algorithms of Section 5.4, we can rewrite an element given on the basis $(x_m^i)_{0 \le i < m}$ on the basis $(x_{m'}^i x_{\ell^k}^j)_{0 \le i < m', 0 \le j < \ell^k}$. Using an algorithm for embeddings in the ℓ-adic tower, we can then embed on the basis $(x_{m'}^i x_{\ell^{k+1}}^j)_{0 \le i < m', 0 \le j < \ell^{k+1}}$; applying our isomorphism algorithm, we end up on the basis $(x_{m\ell}^i)_{0 \le i < m\ell}$, since $x_{m\ell} = x_{m'} x_{\ell^{k+1}}$.

Further operations. Without entering into details, let us mention that further operations are feasible, in the same spirit as the embedding algorithm we just described.

For instance, for arbitrary integers m and n, it is possible to compute the relative minimal polynomial of x_{mn} over $\mathbb{F}_p(x_m)$; it is obtained as a composed product, with factors deduced from the decomposition of m and n into primes.

As another example, we can compute $\Phi_{(m,n),(mn)}$, that is, go from the univariate basis $(x_{mn}^i)_{0\le i<mn}$ to the bivariate basis $(x_m^i x_{mn}^j)_{0\le i<m, 0\le j<n}$. This can be used to compute for instance relative traces, norms or minimal polynomials of arbitrary elements of $\mathbb{F}_{p^{mn}}$ over $\mathbb{F}_{p^m}$.

5.6 Implementation

To demonstrate the practicality of our algorithms, we made a C implementation and compared it to various ways of constructing the same fields in Magma. All timings in this section are obtained on an Intel Xeon E5620 CPU at 2.40GHz, using Magma V2.18-12, Flint 2.4.1 and Sage 6.

Our implementation is limited to finite fields of word-sized characteristic. It is based on the C library Flint [24], and we make it available as a Sage module in an experimental fork at `https://github.com/defeo/sage/tree/ff_compositum`. We plan to make it available as a standard Sage module, as well as a separate C library, when the code has stabilized.

Based on the observation that algorithms Embed and Project are simpler than conversion algorithms between monomial and dual bases, we chose to implement a *lazy change of basis* strategy. By this we mean that our Sage module (rather than the C library itself) represents elements on either the monomial or the dual basis, with one representation computed from the other only when needed. For example, two elements of the same field can be summed if both have a monomial or if both have a dual representation. Similarly, two elements can be multiplied using standard multiplication if both have a monomial representation, or using transposed multiplication if one of the two has a monomial representation. In all other cases, the required representation is computed and stored when the user input prompts it. To implement this strategy efficiently, our Sage module is written in the compiled language Cython.

We focus our benchmarks on the setting of Section 5.4: P and Q are two irreducible polynomials of coprime degrees m and n, and $R = P \odot Q$. We fix the base field $\mathbb{F}_p$ and make m and n grow together with $n = m + 1$. We measure the time to compute R, to apply the algorithms Embed, Phi1, etc., and to compute the changes of bases. We noticed no major difference between different characteristics, so we chose $p = 5$ for our demonstration. As shown in Figure 5.1, the dominating phase is the computation of R (line labeled R). Surprisingly, transposed modular multiplication is slightly faster than ordinary modular multiplication. The cost of Embed is about the same as that of multiplication, while DualToMonomial is about 50% slower. Project and MonomialToDual have, respectively, similar performances (only slightly faster) hence they are not reported on the graph. This justifies our design choice of *lazy change of basis*.

Unsurprisingly, the isomorphism algorithms take significantly more time than the computation of R; for our choices of degrees, Phi2 is asymptotically faster than Phi1 and the crossover between them happens around $m = 70$.

We compare our implementation to four different strategies available in Magma. For each of them we measure the time to construct the finite fields and embedding data, as well as the time to do operations equivalent to Embed, resp. inverse isomorphism.

Figure 5.2 reports on the following experiments. In irred, we supply directly P, Q and R to Magma's finite field constructor, then we call the `Embed` routine to compute the embedding data. In P R, we use Magma's default constructor to compute P and R (Magma chooses its own polynomials), then we call the `Embed` routine to compute the embedding. In P Q, we use Magma's default constructor to compute P and Q (Magma chooses its own polynomials), then use the

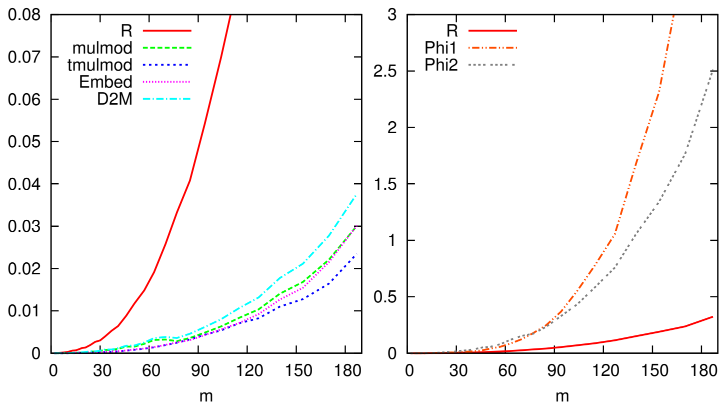


Figure 5.1: Timings in seconds, $p = 5$, $n = m + 1$

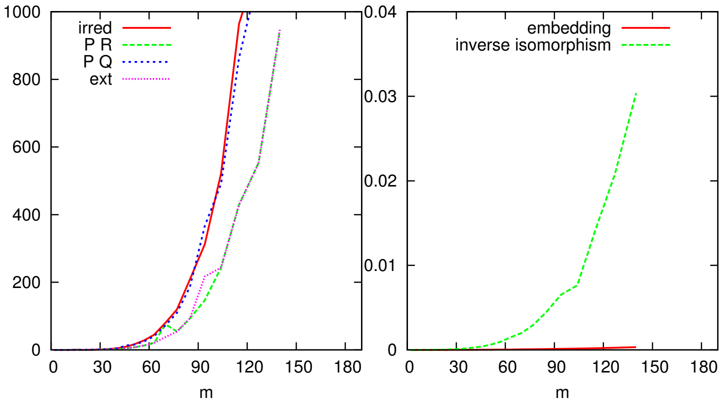


Figure 5.2: Magma timings in seconds, $p = 5$, $n = m + 1$

`CommonOverfield` routine to compute R, then `Embed` to compute the embedding data. In `ext`, we use Magma's default constructor to compute P, then the `ext` operator to compute an extension of degree n of $\mathbb{F}_p[x]/\langle P\rangle$ (Magma chooses its own polynomials).

Timings for constructing the extension and the embedding vary from one method to the other; once this is done, timings for applying embeddings or (inverse) isomorphisms are the same across these methods.

The Magma implementation cannot construct the embedding data in large cases ($m = 150$) in less than 1000 seconds, while our code takes a few seconds. Once the embedding data is known, Magma can apply the embeddings or isomorphisms extremely fast; in our case, one may do the same, using our algorithms to compute the matrices of Φ and Φ^{-1}, when precomputation time and memory are not a concern.

Acknowledgements. We would like to thank the referees for their insightful remarks. Part of this work was financed by NSERC, the CRC program and the ANR project ECLIPSES (ANR-09-VERS-018).

Bibliography

[1] B. Allombert. Explicit computation of isomorphisms between finite fields. *Finite Fields Appl.*, 8(3):332 – 342, 2002.

[2] E. R. Berlekamp. Bit-serial Reed-Solomon encoders. *IEEE Trans. Inf. Theory*, 28(6):869–874, 1982.

[3] W. Bosma, J. Cannon, and C. Playoust. The MAGMA algebra system I: the user language. *J. Symb. Comput.*, 24(3-4):235–265, 1997.

[4] W. Bosma, J. Cannon, and A. Steel. Lattices of compatibly embedded finite fields. *J. Symb. Comput.*, 24(3-4):351–369, 1997.

[5] A. Bostan. *Algorithmes rapides pour les polynômes, séries formelles et matrices*, volume 1 of *Les cours du CIRM*. 2010.

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

[7] A. Bostan, L. González-Vega, H. Perdry, and É. Schost. From Newton sums to coefficients: complexity issues in characteristic p. In *MEGA'05*, 2005.

[8] A. Bostan, G. Lecerf, and É. Schost. Tellegen's principle into practice. In *ISSAC'03*, pages 37–44. ACM, 2003.

[9] N. Bourbaki. *Éléments de mathématique*. Springer, 2007. Algèbre. Chapitre 9.

[10] J. V. Brawley and L. Carlitz. Irreducibles and the composed product for polynomials over a finite field. *Discrete Math.*, 65(2):115–139, 1987.

[11] R. P. Brent and H.-T. Kung. Fast algorithms for manipulating formal power series. *Journal of the ACM*, 25(4):581–595, 1978.

[12] P. Bürgisser, M. Clausen, and M. A. Shokrollahi. *Algebraic Complexity Theory*. Springer, February 1997.

[13] D. G. Cantor. On arithmetical algorithms over finite fields. *J. Combin. Theory Ser. A*, 50(2):285–300, 1989.

[14] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9(3):251–280, 1990.

[15] J.-M. Couveignes. Isomorphisms between Artin-Schreier towers. *Math. Comp.*, 69(232):1625–1631, 2000.

[16] J.-M. Couveignes and R. Lercier. Fast construction of irreducible polynomials over finite fields. *Israel J. Math.*, 194(1):77–105, 2013.

[17] D. A. Cox, J. Little, and D. O'Shea. *Using Algebraic Geometry*. Springer-Verlag, 2005.

[18] L. De Feo, J. Doliskani, and É. Schost. Fast algorithms for ℓ-adic towers over finite fields. In *ISSAC'13*, pages 165–172. ACM, 2013.

[19] L. De Feo and É. Schost. Fast arithmetics in Artin-Schreier towers over finite fields. *J. Symb. Comput.*, 47(7):771–792, 2012.

[20] Luca De Feo. *Algorithmes Rapides pour les Tours de Corps Finis et les Isogénies*. PhD thesis, École Polytechnique X, December 2010.

[21] J. Doliskani and É. Schost. Computing in degree 2^k-extensions of finite fields of odd characteristic. *Des. Codes Cryptogr.*, to appear.

[22] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, New York, NY, USA, 1999.

[23] G. Hanrot, M. Quercia, and P. Zimmermann. The middle product algorithm I. *Appl. Algebra Engrg. Comm. Comput.*, 14(6):415–438, 2004.

[24] William Hart. Fast library for number theory: an introduction. *Mathematical Software-ICMS 2010*, pages 88–91, 2010.

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

[26] E. Kunz. *Kähler differentials*. Friedr. Vieweg & Sohn, 1986.

[27] R. Lebreton, E. Mehrabi, and É. Schost. On the complexity of solving bivariate systems: The case of non-singular solutions. In *ISSAC'13*, pages 251–258. ACM, 2013.

[28] H. W. Lenstra Jr. Finding isomorphisms between finite fields. *Math. Comp.*, 56(193):329–347, 1991.

[29] H. W. Lenstra Jr. and B. De Smit. Standard models for finite fields: the definition, 2008.

[30] Xin Li, Marc Moreno Maza, and Éric Schost. Fast arithmetic for triangular sets: from theory to practice. In *ISSAC '07: Proceedings of the 2007 international symposium on Symbolic and algebraic computation*, pages 269–276, New York, NY, USA, 2007. ACM.

[31] The PARI Group, Bordeaux. *PARI/GP, version* `2.7.0`, 2014.

[32] A. Poteaux and É. Schost. Modular composition modulo triangular sets and applications. *Comput. Complexity*, 22(3):463–516, 2013.

[33] A. Poteaux and É. Schost. On the complexity of computing with zero-dimensional triangular sets. *J. Symb. Comput.*, 50:110–138, 2013.

[34] F. Rouillier. Solving Zero-Dimensional systems through the Rational Univariate Representation. *Appl. Algebra Engrg. Comm. Comput.*, 9(5):433–461, 1999.

[35] V. Shoup. New algorithms for finding irreducible polynomials over finite fields. *Math. Comp.*, 54:435–447, 1990.

[36] V. Shoup. Fast construction of irreducible polynomials over finite fields. *J. Symb. Comput.*, 17(5):371–391, 1994.

[37] Victor Shoup. NTL: A library for doing number theory. `http://www.shoup.net/ntl`.

[38] Victor Shoup. Efficient computation of minimal polynomials in algebraic extensions of finite fields. In *ISSAC'99*, pages 53–58. ACM, 1999.

[39] William A. Stein and Others. *Sage Mathematics Software (Version 5.5).* The Sage Development Team, 2013.

[40] V. Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *STOC'12*, pages 887–898. ACM, 2012.

[41] J. von zur Gathen and V. Shoup. Computing Frobenius maps and factoring polynomials. *Comput. Complexity*, 2:187–224, 1992.

Conclusion

We have presented algorithms to construct and perform computations in algebraic closures of finite fields. Most of our algorithms are quasi-linear in the degree of the extension. Experiments show that our algorithms and implementations, which use monomial representation, are superior to those based on linear algebra. Future directions of this work would include: (i) Designing more efficient algorithms for computing isomorphisms, and (ii) Making the construction of towers quasi-linear in all cases.

Appendix A

Finite Fields

In this appendix, we briefly review the theory of finite fields[1]. A field K is a set equipped with two operations $+ : K \times K \to K$, and $\times : K \times K \to K$ called addition and multiplication. The following conditions are imposed for all $a, b, c \in K$:

1. Associativity: $a \times (b \times c) = (a \times b) \times c, a + (b + c) = (a + b) + c$
2. Commutativity: $a \times b = b \times a, a + b = b + a$
3. Identity: there exist elements denoted by $0, 1$ in K such that $a \times 1 = a$, and $a + 0 = a$.
4. Inverse: for $a \neq 0$, there exist $-a, a^{-1} \in K$ such that $a + (-a) = 0$, and $a \times a^{-1} = 1$.
5. Distributivity: $a \times (b + c) = a \times b + a \times c$.

We usually use familiar notations for the above two operations, e.g. the multiplication symbol is often skipped. It is apparent from the above conditions that the elements $K^* = K \backslash 0$ form a multiplicative group. The multiplicative order of an element $a \in K$ is a the smallest positive integer n (if exists) such that $a^n = 1$. The characteristic of a field K, denoted by char(K), is the smallest positive integer n (if exists) such that

$$\underbrace{1 + \cdots + 1}_{n \text{ times}} = 0.$$

If such integer does not exists, the characteristic is ∞. For any field K, char(K) is either ∞ or a prime number. A homomorphism $\varphi : E \to F$ of fields is a homomorphism of E, F considered as rings. It preserves addition and multiplication. More precisely, $\varphi(ab) = \varphi(a)\varphi(b)$, and $\varphi(a + b) = \varphi(a) + \varphi(b)$ for $a, b \in E$.

Polynomial ring. Given a ring R, the set of univariate polynomial with coefficients in R is a ring denoted by $R[X]$. The ring of bivariate polynomials is defined by $R[X, Y] = R[X][Y]$, and polynomial rings with higher number of variables are defined inductively. When R is a field K then every ideal of $K[X]$ is of the form $\langle f \rangle$, i.e. it is generated by a polynomial $f \in K[X]$. A polynomial $f \in K[X]$ is irreducible if it cannot be written as $f = gh$ with $\deg g, \deg h > 0$. The ideal $\langle f \rangle$ is prime if and only if the polynomial f is irreducible. In that case the quotient

[1] We assume the reader has a very basic knowledge of some algebraic objects like Groups, Rings, and Ideals.

$K[X]/\langle f\rangle$ is a field. An element a in some extension F of K is a root of $f\in K[X]$ if $(X-a)$ is a factor of f in $F[X]$ or equivalently if $f(a)=0$. Each term of a polynomial is called a monomial. The monomial of the highest degree is called the leading term, and it coefficient is called the leading coefficient. A monic polynomial is a polynomial with leading coefficient equal to 1. Over a field every polynomial can be made monic by multiplying it by the inverse of its leading coefficient.

A ***finite field*** is a field with a finite number of elements. The most familiar finite fields are the prime fields. Given a prime number p, a prime finite field, denoted by $\mathbb{F}_p$, is a field consisting of numbers $\{0,1,\dots,p-1\}$. The operations are done module p, and p is called the modulus. Any field K with char$(K)=p$ contains a copy of $\mathbb{F}_p$. In other words, K is a vector space over $\mathbb{F}_p$. This means the cardinality of a finite field is always a prime power, namely $p^{[K:\mathbb{F}_p]}$, where $[K:\mathbb{F}_p]$ is the degree of the extension $\mathbb{F}_p\subseteq K$.

A.1 Basic properties

Assume a finite field F has a subfield $E\subseteq F$ of size q. If the degree of the extension $E\subseteq F$ is n then F has q^n elements. Indeed, the elements of F can be represented as unique sums $a_1x_1+a_2x_2+\cdots+a_nx_n$ where $a_i\in E$ and $\{x_i\}_i$ is a basis of E over F as a vector space. We shall denote a finite field of size q, where $q=p^n$ is a prime power, by $\mathbb{F}_q$. Every element $a\in\mathbb{F}_q$ satisfies $a^q=a$, since the multiplicative group $\mathbb{F}_q^*$ has size $q-1$. In other words, every element of $\mathbb{F}_q$ is a root of the polynomial $g(X)=X^q-X\in\mathbb{F}_p[X]$. We say that $\mathbb{F}_q$ is a splitting field of $g(X)$. In general, the a splitting field of a polynomial f over a field K is a the smallest field $L\supseteq K$ containing all the roots of f. Splitting fields always exists, and they are unique up to isomorphisms. This yields the following result.

❀ *For any given prime p and positive integer n there exists a finite field of size $q=p^n$. Any two such finite field are isomorphic to the splitting field of X^q-X over $\mathbb{F}_p$.*

Therefore, we can always talk about *the* finite field of a given size. Since char$(\mathbb{F}_q)=p$ it is easy to check that $(a+b)^p=a^p+b^p$. This yields the famous automorphism

$$\begin{array}{rccc}\phi_p: & \mathbb{F}_q & \to & \mathbb{F}_q\\ & a & \mapsto & a^p\end{array}$$

called ***Frobenius automorphism***. Different powers of the Frobenius are defined by composition, e.g. $\phi^2=\phi\circ\phi$. In fact we have a cyclic group $G=\{1,\phi,\phi^2,\dots,\phi^{n-1}\}$ of order n. This group is called the Galois group of the extension $\mathbb{F}_p\subseteq\mathbb{F}_q$, and is denoted by Gal$(\mathbb{F}_q/\mathbb{F}_p)$. For every element $\sigma\in G$ there is a subset $F\subseteq\mathbb{F}_q$ such that $\sigma(a)=a$ for all $a\in F$. One can check that F is a field. We call F the fixed field of σ. Similarly, every subgroup of G has a fixed field. In fact, it can be shown that there is a one-to-one correspondence between the subgroups of G and subfields of $\mathbb{F}_q$. The subgroups of G are unique and correspond to the divisors of n. This translate, via the above correspondence, to the following result.

❀ *For ever divisor $m\mid n$ there is exactly one subfield of $\mathbb{F}_q$ of size p^m. Conversely, every subfield of $\mathbb{F}_q$ is of size p^m with $m\mid n$.*

Let K be an arbitrary field, and let $G \subseteq K^*$ be a finite subgroup of size n. Let $a \in G$ be an element with maximal order m. Then the order of every other element divides m. In fact, if $m_1 > 1$ is the order of an element $b \in G$, and m_1 is coprime to m then ab has order $m_1 m > m$ which contradict the assumption of maximality of m. Therefore, every element of G is a root of $g(X) = X^m - 1$. Since, over a field, $g(X)$ can have at most m roots we have $m = n$. So we have found a generator for G, hence G is cyclic.

❀ *The multiplicative group $\mathbb{F}_q^*$ is cyclic.*

A generator of the group $\mathbb{F}_q^*$ is called a primitive element of $\mathbb{F}_q$. If $a \in \mathbb{F}_q$ is a primitive element then a^r is also a primitive element for all r coprime to $q-1$. Therefore, $\mathbb{F}_q$ has exactly $\varphi(q-1)$ primitive elements, where φ is the Euler's totient function.

A.2 Irreducible polynomials

Let $F \subseteq E$ be an extension of finite fields. We say that the extension is algebraic if for every element $a \in E$ there exists a polynomial g over F such that $g(a) = 0$. We define the ***minimal polynomial*** of an element $a \in E$ to be a monic polynomial g over F of minimal degree with $g(a) = 0$. The minimality condition on the degree implies that minimal polynomials are always irreducible. Indeed, if $g = h_1 h_2$ with $\deg h_1, \deg h_2 > 0$ then $g(a) = h_1(a)h_2(a) = 0$, and hence say $h_1(a) = 0$ with $\deg h_1 < \deg g$ which is a contradiction.
Another way of introducing minimal polynomials is as follows. As mentioned before, every ideal in $F[X]$ can be written as $\langle f \rangle$ for some $f \in F[X]$. This is, in fact, the result of $F[X]$ being an Euclidean domain; i.e. for every $a, b \in F[X]$ with $g \neq 0$, there are $q, r \in F[X]$ such that $a = bq + r$ and either $r = 0$ or $\deg r < \deg b$. Let $I \subset F[X]$ be an ideal, and let $f \in I$ be a polynomial with lowest degree. We can assume that f is monic. Given $g \in F[X]$ we can write $g = fq + r$. If $r \neq 0$ then r then $r = g = fq \in I$ and it has a lower degree than f a contradiction. Therefore, f divides every polynomial in I, and hence $I = \langle f \rangle$. Now, given $a \in E$ let I be the set of all $g \in F[X]$ such that $g(a) = 0$. One readily checks that I is an ideal. Write $I = \langle f \rangle$, and define f as the minimal polynomial of a. From this we see that minimal polynomials are unique. It is easy to check that the above two definitions are equivalent. The latter yields the following result.

❀ *Let $F \subseteq E$ be finite field extensions, and let $f \in F[X]$ be the minimal polynomial of an element $a \in E$. Then for any $g \in F[X]$ we have $g(a) = 0$ if and only if $f \mid g$.*

One of the interesting polynomials over $\mathbb{F}_q$ is $g(X) = X^{q^r} - X$ for a given $r > 0$. Suppose that an irreducible polynomial $f \in \mathbb{F}_q[X]$ of degree m divides this polynomial. The the two polynomials have a common root a in the splitting field of f over $\mathbb{F}_q$. Since $a^{q^r} = a$ we have the extensions $\mathbb{F}_q \subseteq \mathbb{F}_{q^m} \subseteq \mathbb{F}_q^r$ hence $m \mid r$. Conversely, if $m \mid r$ then we have the above extensions and $\mathbb{F}_{q^m}$ is the splitting field of f. So f and g have a common root $a \in \mathbb{F}_{q^r}$. But f is the minimal polynomial of a over $\mathbb{F}_q$ hence $f \mid g$. So we have proved the following.

❀ *Let f be an irreducible polynomial of degree m over $\mathbb{F}_q$, and let $g(X) = X^{q^r} - X$. Then $f \mid g$ if and only if $m \mid r$.*

The above result say that for a given $r > 0$, g is a the product of all irreducible polynomials who's degrees divide r. An immediate application of this result is testing for irreducibility. A polynomial f of degree m is irreducible if and only if

i . f divides $X^{q^m} - X$,

ii . $\gcd(X^{q^{m/t}} - X, f) = 1$ for all prime divisors t of m.

An interesting observation about irreducible polynomials over finite fields is that any extension containing one root of an irreducible polynomial contains all the other roots as well. More precisely, if $f(X) = X^m + a_{m-1}X^{m-1} + \cdots + a_0$ is an irreducible polynomial over $\mathbb{F}_q$, and $b \in F_{q^m}$ is a root of f then we have $b^m + a_{m-1}b^{m-1} + \cdots + a_0 = 0$. Raising both sides to the power of q^i for any $1 \leq i \leq m-1$ we get $(b^{q^i})^m + a_{m-1}(b^{q^i})^{m-1} + \cdots + a_0 = 0$. One checks that the distinct elements $b, b^q, \ldots, b^{q^{m-1}}$ are all the roots of f. These elements are called the conjugates of b. More generally, given an extension $\mathbb{F}_q \subseteq \mathbb{F}_{q^m}$, and an element $a \in \mathbb{F}_{q^m}$ we define the conjugates of a with respect to $\mathbb{F}_q$ as $a, a^q, \ldots, a^{q^{m-1}}$. The terminology comes from the action of the elements of $\mathrm{Gal}(\mathbb{F}_{q^m}/\mathbb{F}_q) = \{1, \sigma, \sigma^2, \ldots, \sigma^{m-1}\}$, where $\sigma^i(x) = x^{q^i}$, on a. From the above we also see that $\mathbb{F}_{q^m}$ is the splitting field of f over $\mathbb{F}_q$. Therefore, two irreducible polynomials of the same degree have isomorphic splitting fields.

From the beginning we have implicitly assumed that the Galois group $\mathrm{Gal}(\mathbb{F}_{q^m}/\mathbb{F}_q)$, which is defined to be the group of all automorphisms $\alpha : \mathbb{F}_{q^m} \to \mathbb{F}_{q^m}$ over $\mathbb{F}_q$, consists only of σ^i defined above. This is always the case for finite fields. In fact, let α be an arbitrary automorphism of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$. Also let β be a primitive element of $\mathbb{F}_{q^m}$, and let f be its minimal polynomial of $\mathbb{F}_q$. So $0 = \alpha(f(\beta)) = f(\alpha(\beta))$ hence $\alpha(\beta)$ is also a root of f. Since all other roots of f are conjugates of β we must have $\alpha(\beta) = \beta^{q^i}$ for some $0 \leq i \leq m-1$. Also since β is a primitive element we have $\alpha(a) = a^{q^i}$ for all $a \in \mathbb{F}_{q^m}$.

Cyclotomic polynomials. Let r be a positive integer such that $r \mid q-1$. Then there is an element $\zeta \in \mathbb{F}_q$ of order r, namely $g^{(q-1)/r}$ for some generator $g \in \mathbb{F}_q^*$. The element ζ is called a primitive rth root of unity. Also for all $1 \leq i < r$ coprime to r, ζ^i is also a primitive rth root of unity. Define the rth Cyclotomic polynomial as

$$\Phi_r(X) = \prod_{\substack{1 \leq i < r \\ \gcd(i,r)=1}} (X - \zeta^i).$$

we obviously have $\deg \Phi_r = \phi(r)$ where ϕ is the Euler function. The polynomial Φ_r is square-free by definition. Let f be the minimal polynomial of ζ over $\mathbb{F}_p$, and let d be the order of p in the multiplicative group $\mathbb{Z}/r\mathbb{Z}$. Then we know that $d \mid \phi(r)$ by group theory. As before, ζ^{p^i} is also a root of f for all $0 \leq i < \phi(r)$. But only d of these elements are distinct, namely $A = \{\zeta, \zeta^{p^1}, \ldots, \zeta^{p^{d-1}}\}$. So f has degree d. We can repeat the same process for the minimal polynomial of an element ζ^{p^i} not in A, and append the next set of distinct powers to A, and so on. All these minimal polynomials divide Φ_r. This yields the following.

❀ *Let r be a positive integer such that $r \mid q-1$, and let d be the order of q in $\mathbb{Z}/r\mathbb{Z}$. Then Φ_r factors into $\phi(r)/d$ irreducible polynomials of the same degree d.*

A.3 Traces and Norms

Let $F=\mathbb{F}_q$, and $E=\mathbb{F}_{q^m}$ be an extension of F. The trace map from E to F is defined as

$$\begin{array}{rccl} \mathrm{tr}_{E/F}: & E & \to & F \\ & a & \mapsto & a+a^q+\cdots+a^{q^{m-1}}. \end{array}$$

So the trace of an element is simply the sum of its conjugates. One hidden fact in the above definition is that the image of the trace is actually contained in F. This is a direct consequence of the fact that trace is fixed by all $\sigma \in \mathrm{Gal}(E/F)$. Indeed,

$$\begin{aligned} \mathrm{tr}_{E/F}(a)^q &= (a+a^q+\cdots+a^{q^{m-1}})^q \\ &= a^q+\cdots+a^{q^{m-1}}+a \\ &= \mathrm{tr}_{E/F}(a). \end{aligned}$$

One can easily check that $\mathrm{tr}_{E/F}$ is a linear map over F, or an F-linear map, considering both E,F as vector spaces over F; i.e.

$$\mathrm{tr}_{E/F}(a\alpha+b\beta)=a\,\mathrm{tr}_{E/F}(\alpha)+b\,\mathrm{tr}_{E/F}(\beta) \quad a,b\in F \text{ and } \alpha,\beta\in E.$$

This mean $\mathrm{tr}_{E/F}(a)=ma$ for all $a\in F$. An element $a\in E$ is in the kernel K of the trace map if it is a root of the polynomial $X+X^q+\cdots+X^{q^{m-1}}$. But this polynomial has at most q^{m-1} roots. So $\#K\le q^{m-1}$ hence the image of $\mathrm{tr}_{E/F}$ has size larger than q. Therefore, the trace map is surjective. We saw before that every automorphism of $\mathbb{F}_{q^m}$ is of the form $x\mapsto x^{q^i}$ for some $0\le i\le m-1$. This extends to the case of trace maps as follows. Define $\ell_b(a)=\mathrm{tr}_{E/F}(ab)$ for $b\in E$ and all $a\in E$. If $a\ne a'$ then $\ell_a-\ell_{a'}=\mathrm{tr}_{E/F}(ab)-\mathrm{tr}_{E/F}(ab)=\mathrm{tr}_{E/F}((a-a')b)$ which is not zero for some b as the trace in onto. Therefore, $\ell_a\ne\ell_{a'}$. Also there are only a finite number of F-linear maps $E\to F$. In fact, every such linear map is determined by assigning elements of a given basis of E to elements of F. So there are q^m of such maps. But there are the same number of maps ℓ_b as well. Therefore, every F-linear map $E\to F$ is of the form ℓ_b for some $b\in E$.
Let E,F be as above. The norm map from E to F is defined as

$$\begin{array}{rccl} \mathrm{N}_{E/F}: & E & \to & F \\ & a & \mapsto & aa^q\cdots a^{q^{m-1}}=a^{(q^m-1)/(q-1)}. \end{array}$$

Again the image of $\mathrm{N}_{E/F}$ is always in F. From the definition we have

$$\mathrm{N}_{E/F}(ab)=\mathrm{N}_{E/F}(a)\mathrm{N}_{E/F}(b) \quad a,b\in E.$$

This means that norm is a homomorphism $E^*\to F^*$ of groups. We also have $\mathrm{N}_{E/F}(a)=a^m$ for all $a\in F$. Like trace, norm is also onto: the kernel of $\mathrm{N}_{E/F}$ is a subset of the roots of the polynomial $X^{(q^m-1)/(q-1)}$. So the kernel has size smaller than $(q^m-1)/(q-1)$ hence the image of the map has size larger than $q-1$. The norm and trace maps are both transitive in the following sense.

❀ *For a chain of extensions $F\subset K\subset E$ of finite fields*

$$\mathrm{tr}_{E/F}(a)=\mathrm{tr}_{K/F}(\mathrm{tr}_{E/K}(a)), \quad \mathrm{N}_{E/F}(a)=\mathrm{N}_{E/K}(\mathrm{N}_{K/F}(a))$$

for all $a\in E$.

A.4 Algebraic closures

In this section, we discuss the basic concepts of algebraic closures and their construction. We will also discuss our computational approach to dealing with algebraic closures of finite fields.

A field L is said to be algebraically closed if every non-constant polynomial in $L[X]$ has a root in L. This is equivalent to saying that every non-constant polynomial splits into linear factors over L. An ***algebraic closure*** of a field K, denoted by $\overline{K}$, is an algebraic extension of K that is algebraically closed.

Given a field K one can build an algebraically closed filed containing K as follows. We first build a field K_1 such that every polynomial in $K[X]$ has a root in K_1. Let $\mathscr{S}$ be the set of all polynomials $f \in K[X]$, and let $\mathscr{X}$ be the set of variables $\{X_f\}_{f\in\mathscr{S}}$. So we have introduced a variable for each polynomial. Now form the ring $K[\mathscr{X}]$, and let $I \subset K[\mathscr{X}]$ be the ideal generated by all the polynomials $f(X_f)$. We claim that I is not the unit ideal. If it is, then there is a finite linear combination

$$g_1 f_1(X_{f_1}) + \cdots + g_n f_n(X_{f_n}) = 1, \qquad g_i \in K[\mathscr{X}].$$

This equation involves only a finite number of variables, say $X_{f_1}, \ldots, X_{f_N}$. Therefore, rewriting the equation gives

$$\sum_{i=1}^{n} g_i(X_{f_1}, \ldots, X_{f_N}) f_i(X_{f_i}) = 1.$$

There exists a finite extension E of K in which each f_i has a root. Let $a_i \in E$ be a root of f_i. Then substituting a_i for X_{f_i} in the above equation will give $0 = 1$ in E, which is a contradiction. So a simple application of Zorn's lemma gives $I \subseteq \mathfrak{m}$ for some maximal ideal $\mathfrak{m}$ of $K[\mathscr{X}]$. So $K_1 = K[\mathscr{X}]/\mathfrak{m}$ is a field containing K. Also every polynomial $f \in K[X]$ has a root in K_1 by construction. Repeating the same process for K_1, and so on, we obtain a tower

$$K = K_0 \subseteq K_1 \subseteq K_2 \subseteq \cdots$$

in which every non-constant polynomial in $K_n[X]$ has a root in K_{n+1} for all $n \geq 0$. Now define K_∞ to be the union of all these extensions. One can easily check that K_∞ is a field. Let $f \in K_\infty[X]$. Then the coefficients of f are in K_n for a large enough n. So f has a root in $K_{n+1} \subseteq K_\infty$, hence K_∞ is algebraically closed.

So given a field K there is an algebraically closed extension $K \subseteq E$. Define $F \subseteq E$ as the union of all subfields of E that are algebraic over K. It is easy to check that F is algebraic over K and it is algebraically closed. Therefore, F is an algebraic closure of K. It can be shown that algebraic closures are unique up to isomorphism of fields.

Finite fields. It is a simpler and more intuitive situation for algebraic closures over finite fields. Starting from $\mathbb{F}_p$, we know that every irreducible polynomial $f \in \mathbb{F}_p[X]$ of degree n has a root in $\mathbb{F}_{p^n}$. So adapting the above general construction of adding roots of polynomials we see that we only need to consider extensions $\mathbb{F}_{p^2}, \mathbb{F}_{p^3}, \mathbb{F}_{p^4}, \ldots$. From previous sections we know that $\mathbb{F}_{p^m} \subseteq \mathbb{F}_{p^n}$ if and only if $m \mid n$. So, there are inclusions $\mathbb{F}_{p^{n_1}} \subseteq \mathbb{F}_{p^{n_1 n_2}}$ and $\mathbb{F}_{p^{n_2}} \subseteq \mathbb{F}_{p^{n_1 n_2}}$ for every $n_1, n_2 > 0$. So the above set of finite fields is partially ordered by inclusion, and we can talk about the union of any two finite fields. Then we have

$$\overline{\mathbb{F}_p} = \bigcup_{i\geq 1} \mathbb{F}_{p^i}.$$

In fact, let $K = \cup_{i\geq 1}\mathbb{F}_{p^i}$, and let $a \in K$. Then $a \in \mathbb{F}_{p^n}$ for some n, hence a is algebraic over $\mathbb{F}_p$. Also every irreducible polynomial $f \in K[X]$ has coefficients in $\mathbb{F}_{p^n}$ for a large enough n. Then f has a root in $\mathbb{F}_{p^n}[X]/\langle f\rangle \cong \mathbb{F}_{p^{mn}} \subseteq K$ where $m = \deg f$.

VITA

Javad Doliskani

Ontario Research Center for Computer Algebra (ORCCA)
Department of Computer Science, University of Western Ontario
London, Ontario, Canada, N6A 5B7

Education

2003-2008	Tarbiat Moallem University, Tehran, Tehran, Iran, Software Engineering, B.S.
2009-2011	University of Western Ontario, London, Ontario, Canada, Computer Algebra, MSc.
2011-2015	University of Western Ontario, London, Ontario, Canada, Computer Algebra, PhD.