Fast Arithmetics in Artin-Schreier Towers over Finite Fields

Luca De Feo
LIX, École Polytechnique
Palaiseau, France
luca.defeo@polytechnique.edu

Éric Schost
ORCCA and CSD
The University of Western Ontario, London, ON
eschost@uwo.ca

ABSTRACT

An *Artin-Schreier tower* over the finite field $\mathbb{F}_p$ is a tower of field extensions generated by polynomials of the form $X^p - X - \alpha$. Following Cantor and Couveignes, we give algorithms with quasi-linear time complexity for arithmetic operations in such towers. As an application, we present an implementation of Couveignes' algorithm for computing isogenies between elliptic curves using the p-torsion.

Categories and Subject Descriptors:
I.1.2 [**Symbolic and Algebraic Manipulation**]: Algorithms – *Algebraic Algorithms*

General Terms: Algorithms, Theory

Keywords: Algorithms, complexity, Artin-Schreier

1. INTRODUCTION

Definitions. If $\mathbb{U}$ is a field of characteristic p, polynomials of the form $P = X^p - X - \alpha$ with $\alpha \in \mathbb{U}$ are called *Artin-Schreier polynomials*; a field extension $\mathbb{U}'/\mathbb{U}$ is *Artin-Schreier* if it is of the form $\mathbb{U}' = \mathbb{U}[X]/P$, with P an Artin-Schreier polynomial.

An *Artin-Schreier tower* of height k is a sequence of Artin-Schreier extensions $\mathbb{U}_i/\mathbb{U}_{i-1}$, for $1 \leqslant i \leqslant k$; it is denoted by $(\mathbb{U}_0, \ldots, \mathbb{U}_k)$. In what follows, we only consider extensions of finite degree over $\mathbb{F}_p$. Thus, $\mathbb{U}_i$ is of degree p^i over $\mathbb{U}_0$, and of degree $p^i d$ over $\mathbb{F}_p$, with $d = [\mathbb{U}_0 : \mathbb{F}_p]$.

The importance of this concept comes from the fact that all Galois extensions of degree p are Artin-Schreier. As such, they arise frequently, e.g., in number theory (for instance, when computing p^k-torsion groups of Abelian varieties over $\mathbb{F}_p$). The need for fast arithmetics in these towers is motivated in particular by applications to isogeny computation and point-counting in cryptology, as in [7].

Our contribution. We give fast algorithms for arithmetic operations in Artin-Schreier towers. Prior results for this task are due to Cantor [6] and Couveignes [8]. However, the algorithms of [8] need as a prerequisite a fast multiplication algorithm in some towers of a special kind (called "Cantor towers" in [8]). Such an algorithm is unfortunately not in the literature, making the results of [8] non practical.

This paper fills the gap. Technically, our main algorithmic contribution is a fast change-of-basis algorithm; it makes it possible to obtain fast multiplication routines, and by extension completely explicit versions of all algorithms of [8]. Along the way, we also extend constructions of Cantor to the case of a general finite base field $\mathbb{U}_0$, where Cantor had $\mathbb{U}_0 = \mathbb{F}_p$. As an application, we put to practice Couveignes' isogeny computation algorithm [7].

Complexity notation. We count time complexity in number of operations in $\mathbb{F}_p$. Then, notation being as before, optimal algorithms in $\mathbb{U}_k$ would have complexity $O(p^k d)$; most of our results are (up to logarithmic factors) of the form $O(p^{k+\alpha} d^{1+\beta})$, for small constants α, β such as $0, 1, 2$ or 3.

Many algorithms below rely on fast multiplication; thus, we let $\mathsf{M} : \mathbb{N} \to \mathbb{N}$ be a *multiplication function*, such that polynomials of degree less than n can be multiplied in $\mathsf{M}(n)$ operations, under the conditions of [11, Ch. 8.3]. Typical orders of magnitude for $\mathsf{M}(n)$ are $O(n^{\log_2 3})$ for Karatsuba multiplication or $O(n \log n \log\log n)$ for FFT multiplication. Using fast multiplication, fast algorithms are available for Euclidean division or extended GCD [11, Ch. 9 & 11].

For several operations, different algorithms will be available, and their relative efficiencies can depend on the values of p, d and k. In these situations, we always give details for the case where p is small, since cases such as $p = 2$ or $p = 3$ are especially useful in practice. Some of our algorithms could be slightly improved, but we usually prefer giving the simpler solutions.

Previous work. As said above, this paper builds on former results of Cantor [6] and Couveignes [8, 7]; to our knowledge, prior to this paper, no previous work provided the missing ingredients to put Couveignes' algorithms to practice. Part of Cantor's results were independently discovered by Wang and Zhu [26] and have been extended in another direction (fast polynomial multiplication over arbitrary finite fields) by von zur Gathen and Gerhard [12] and Mateer [20].

Organization of the paper. Section 2 consists in preliminaries: trace computations, duality, basics on Artin-Schreier extensions. In Section 3, we define a specific Artin-Schreier tower, where arithmetic operations will be fast. Our key change-of-basis algorithm for this tower is in Section 4. In Sections 5 and 6, we revisit Couveignes' isomorphism algorithm [8] in our context, giving fast arithmetics for *any* Artin-Schreier tower. Finally, Section 7 gives experimental results obtained by applying our algorithms to Couveignes' isogeny algorithm [7] for elliptic curves.

ISSAC'09, July 28–31, 2009, Seoul, Republic of Korea.

2. PRELIMINARIES

As a general rule, variables and polynomials are in upper case; elements algebraic over $\mathbb{F}_p$ (or some other field, that will be clear from the context) are in lower case.

2.1 Element representation

Let Q_0 be in $\mathbb{F}_p[X_0]$ and let $(G_i)_{0\leqslant i<k}$ be a sequence of polynomials over $\mathbb{F}_p$, with G_i in $\mathbb{F}_p[X_0,\ldots,X_i]$. We say that the sequence $(G_i)_{0\leqslant i<k}$ *defines the tower* $(\mathbb{U}_0,\ldots,\mathbb{U}_k)$ if for $i\geqslant 0$, $\mathbb{U}_i=\mathbb{F}_p[X_0,\ldots,X_i]/K_i$, where K_i is the ideal generated by

$$\left|\begin{array}{l} P_i = X_i^p - X_i - G_{i-1}(X_0,\ldots,X_{i-1}) \\ \quad\vdots \\ P_1 = X_1^p - X_1 - G_0(X_0) \\ Q_0(X_0) \end{array}\right.$$

in $\mathbb{F}_p[X_0,\ldots,X_i]$, and if $\mathbb{U}_i$ is a field. The residue class of X_i (resp. G_i) in $\mathbb{U}_i$, and thus in $\mathbb{U}_{i+1},\ldots$, is written x_i (resp. γ_i), so that we have $x_i^p - x_i = \gamma_{i-1}$.

Finding a suitable $\mathbb{F}_p$-basis to represent elements of a tower $(\mathbb{U}_0,\ldots,\mathbb{U}_k)$ is a crucial question. If $d=\deg(Q_0)$, a natural basis of $\mathbb{U}_i$ is the multivariate basis $\mathbf{B}_i=\{x_0^{e_0}\cdots x_i^{e_i}\}$ with $0\leqslant e_0<d$ and $0\leqslant e_j<p$ for $1\leqslant j\leqslant i$. However, in this basis, we do not have very efficient arithmetic operations, starting from multiplication. See [18] for details.

As a workaround, we introduce the notion of a *primitive tower*, where for all i, x_i generates $\mathbb{U}_i$ over $\mathbb{F}_p$. In this case, we let $Q_i\in\mathbb{F}_p[X]$ be its minimal polynomial, of degree p^id. In a primitive tower, unless otherwise stated, we represent the elements of $\mathbb{U}_i$ on the $\mathbb{F}_p$-basis $\mathbf{C}_i=(1,x_i,\ldots,x_i^{p^id-1})$.

To stress the fact that $v\in\mathbb{U}_i$ is represented on the basis $\mathbf{C}_i$, we write $v\dashv\mathbb{U}_i$. In this basis, additions and subtractions are done in time p^id, multiplications in time $O(\mathsf{M}(p^id))$ [11, Ch. 9] and inversions in time $O(\mathsf{M}(p^id)\log(p^id))$ [11, Ch. 11].

2.2 Trace and pseudotrace

We continue with a few useful facts on traces. Let $\mathbb{U}$ be a field and let $\mathbb{U}'=\mathbb{U}[X]/Q$ be a separable field extension of $\mathbb{U}$, with $\deg(Q)=d$. For $a\in\mathbb{U}'$, the *trace* $\mathrm{Tr}(a)$ is the trace of the $\mathbb{U}$-linear map M_a of multiplication by a in $\mathbb{U}'$.

The trace is a $\mathbb{U}$-linear form; in other words, Tr is in the dual space $\mathbb{U}'^*$ of the $\mathbb{U}$-vector space $\mathbb{U}'$; we write it $\mathrm{Tr}_{\mathbb{U}'/\mathbb{U}}$ when the context requires it. In finite fields, we also have the following well-known properties:

$$\mathrm{Tr}_{\mathbb{F}_{q^n}/\mathbb{F}_q} : a\mapsto \textstyle\sum_{\ell=0}^{n-1} a^{q^\ell}, \qquad (\mathbf{P}_1)$$

$$\mathrm{Tr}_{\mathbb{F}_{q^{mn}}/\mathbb{F}_q} = \mathrm{Tr}_{\mathbb{F}_{q^m}/\mathbb{F}_q}\circ\mathrm{Tr}_{\mathbb{F}_{q^{mn}}/\mathbb{F}_{q^m}}\ . \qquad (\mathbf{P}_2)$$

Besides, if $\mathbb{U}'/\mathbb{U}$ is an Artin-Schreier extension generated by a polynomial Q and x is a root of Q in $\mathbb{U}'$, then

$$\mathrm{Tr}_{\mathbb{U}'/\mathbb{U}}(x^j)=0 \text{ for } j<p-1;\quad \mathrm{Tr}_{\mathbb{U}'/\mathbb{U}}(x^{p-1})=-1. \qquad (\mathbf{P}_3)$$

Following [8], we also use a generalization of the trace. The nth *pseudotrace* of order m is the $\mathbb{F}_{p^m}$-linear operator

$$\mathrm{T}_{(n,m)} : a\mapsto \textstyle\sum_{\ell=0}^{n-1} a^{p^{m\ell}};$$

for $m=1$, we call it the nth pseudotrace and write T_n.

In our context, for $n=[\mathbb{U}_i:\mathbb{U}_j]=p^{i-j}$ and $m=[\mathbb{U}_j:\mathbb{F}_p]=p^jd$, $\mathrm{T}_{(n,m)}(v)$ coincides with $\mathrm{Tr}_{\mathbb{U}_i/\mathbb{U}_j}(v)$ for v in $\mathbb{U}_i$; however $\mathrm{T}_{(n,m)}(v)$ remains defined for v not in $\mathbb{U}_i$, whereas $\mathrm{Tr}_{\mathbb{U}_i/\mathbb{U}_j}(v)$ is not.

2.3 Duality

Finally, we discuss two useful topics related to duality, starting with the transposition of algorithms.

Introduced by Kaltofen and Shoup, the *transposition principle* relates the cost of computing an $\mathbb{F}_p$-linear map $f: V\to W$ to that of computing the transposed map $f^*: W^*\to V^*$. Explicitly, from an algorithm that performs an $r\times s$ matrix-vector product $b\mapsto Mb$, one can deduce an algorithm with the same complexity, up to $O(r+s)$, that performs the transposed product $c\mapsto M^tc$; see [5, 14, 1].

We give here first consequences of this principle, after [24, 25]. Consider a field extension $\mathbb{U}\to\mathbb{U}'=\mathbb{U}[X]/Q$. For w in $\mathbb{U}'$, recall that $M_w:\mathbb{U}'\to\mathbb{U}'$ is the multiplication map $M_w(v)=vw$. Its dual $M_w^*:\mathbb{U}'^*\to\mathbb{U}'^*$ acts on $\ell\in\mathbb{U}'^*$ by $M_w^*(\ell)(v)=\ell(M_w(v))=\ell(vw)$ for v in $\mathbb{U}'$. We prefer to denote the linear form $M_w^*(\ell)$ by $w\cdot\ell$, keeping in mind that $(w\cdot\ell)(v)=\ell(vw)$.

Suppose then that $\mathbf{D}$ is a $\mathbb{U}$-basis of $\mathbb{U}'$, in which we can perform multiplication in time T. Then by the transposition principle, given w on $\mathbf{D}$ and ℓ on the dual basis $\mathbf{D}^*$, we can compute $w\cdot\ell$ on the dual basis $\mathbf{D}^*$ in time $T+O(\deg(Q))$. We will discuss this in more detail in Section 4.

Suppose finally that $\mathbb{U}'$ is separable over $\mathbb{U}$ and that $b\in\mathbb{U}'$ generates $\mathbb{U}'$ over $\mathbb{U}$; given w in $\mathbb{U}'$, we want to find an expression $w=A(b)$, for some $A\in\mathbb{U}[X]$. Hereafter, for $P\in\mathbb{U}[X]$ of degree at most e, we write $\mathrm{rev}_e(P)=X^eP(1/X)\in\mathbb{U}[X]$. Then, we define $\ell=w\cdot\mathrm{Tr}_{\mathbb{U}'/\mathbb{U}}\in\mathbb{U}'^*$ and

$$M=\textstyle\sum_{j<d}\ell(b^j)X^j,\quad N=M\,\mathrm{rev}_d(Q)\bmod X^d. \qquad (1)$$

This construction solves our problem: Theorem 3.1 in [22] shows that $w=A(b)$, with $A=\mathrm{rev}_{d-1}(N)Q'^{-1}\bmod Q$.

We will hereafter denote by $\mathsf{FindParametrization}(b,w)$ a subroutine that computes this polynomial A. If Q is Artin-Schreier, the cost of $\mathsf{FindParametrization}$ is $O(p^2)$ operations $(+,\times)$ in $\mathbb{U}$: finding the requested values of ℓ fits into this bound, by the proof of [24, Th. 4]; the remaining operations are cheaper (and involve no division), since $Q'=-1$ in the Artin-Schreier case.

3. A PRIMITIVE TOWER

Our first task in this section is to describe a specific Artin-Schreier tower where arithmetics will be fast; then, we explain how to construct this tower. This extends results by Cantor [6, Th. 1.2], who dealt with the case $\mathbb{U}_0=\mathbb{F}_p$.

THEOREM 1. *Let $\mathbb{U}_0=\mathbb{F}_p[X_0]/Q_0$, with Q_0 irreducible of degree d, let $x_0=X_0\bmod Q_0$ and assume that $\mathrm{Tr}_{\mathbb{U}_0/\mathbb{F}_p}(x_0)\neq 0$. Let $(G_i)_{0\leqslant i<k}$ be defined by*

$$\begin{cases} G_0 = X_0 & \\ G_1 = X_1 & \text{if } p=2 \text{ and } d \text{ is odd,} \\ G_i = X_i^{2p-1} & \text{in any other case.} \end{cases}$$

Then, $(G_i)_{0\leqslant i<k}$ defines a primitive tower $(\mathbb{U}_0,\ldots,\mathbb{U}_k)$.

As before, for $i\geqslant 1$, let $P_i=X_i^p-X_i-G_{i-1}$ and for $i\geqslant 0$, let K_i be the ideal $\langle Q_0,P_1,\ldots,P_i\rangle$ in $\mathbb{F}_p[X_0,\ldots,X_i]$. Then the theorem says that for $i\geqslant 0$, $\mathbb{U}_i=\mathbb{F}_p[X_0,\ldots,X_i]/K_i$ is a field, and that $x_i=X_i\bmod K_i$ generates it over $\mathbb{F}_p$. Hereafter, recall that we write $\gamma_i=G_i\bmod K_i$. We first prove the case $p\neq 2$; we then indicate the modifications to bring for $p=2$.

LEMMA 2. *For $i \geqslant 0$, $\mathbb{U}_i$ is a field and, for $i \geqslant 1$, $\mathrm{Tr}_{\mathbb{U}_i/\mathbb{U}_{i-1}}(\gamma_i) = -\gamma_{i-1}$.*

PROOF. Induction on i: for $i = 0$, this is true by hypothesis. For $i \geqslant 1$, assuming that $\mathbb{U}_i$ is a field, we prove that $\mathrm{Tr}_{\mathbb{U}_i/\mathbb{F}_p}(\gamma_i) \neq 0$, which, by [19, Th. 2.25], implies that $X_{i+1}^p - X_{i+1} - \gamma_i$ is irreducible in $\mathbb{U}_i[X_{i+1}]$. For $i = 0$, $\mathrm{Tr}_{\mathbb{U}_0/\mathbb{F}_p}(\gamma_0) = \mathrm{Tr}_{\mathbb{U}_0/\mathbb{F}_p}(x_0)$ is non-zero. For $i \geqslant 1$, we know that $\gamma_i = x_i^{2p-1} = x_i^p x_i^{p-1}$, which rewrites

$$(x_i + \gamma_{i-1})x_i^{p-1} = x_i^p + \gamma_{i-1}x_i^{p-1} = \gamma_{i-1} + x_i + \gamma_{i-1}x_i^{p-1}.$$

By $\mathbf{P}_3$, we get $\mathrm{Tr}_{\mathbb{U}_i/\mathbb{U}_{i-1}}(\gamma_i) = -\gamma_{i-1}$ and by $\mathbf{P}_2$, we deduce $\mathrm{Tr}_{\mathbb{U}_i/\mathbb{F}_p}(\gamma_i) = -\mathrm{Tr}_{\mathbb{U}_{i-1}/\mathbb{F}_p}(\gamma_{i-1})$. The induction assumption implies that this is non-zero, and the claim follows. □

LEMMA 3. *For $i \geqslant 0$, γ_i generates $\mathbb{U}_i$ over $\mathbb{F}_p$.*

PROOF. Let $d_i = [\mathbb{F}_p[\gamma_i] : \mathbb{F}_p]$, we want to prove that $d_i = p^i d$. Let $d = p^s r$ with r prime to p, then $p^{i+s} | d_i$; indeed, if it is not the case, $\mathrm{Tr}_{\mathbb{U}_i/\mathbb{F}_p[\gamma_i]}(\gamma_i) = \frac{p^{i+s}r}{d_i}\gamma_i = 0$, which contradicts Lemma 2. Furthermore, $d | d_i$, in fact $\mathrm{Tr}_{\mathbb{U}_i/\mathbb{U}_0}(\gamma_i) \in \mathbb{F}_p[\gamma_i]$, but by Lemma 2 and by $\mathbf{P}_3$, $\mathrm{Tr}_{\mathbb{U}_i/\mathbb{U}_0}(\gamma_i) = (-1)^i\gamma_0$, which generates $\mathbb{U}_0$ by hypothesis. Since $(p^{i+s}, d) = p^s$, $d_i \geqslant p^i d$ and the claim follows. □

The theorem is now an easy consequence of Lemmas 2 and 3 since clearly $\mathbb{F}_p[\gamma_i] \subset \mathbb{F}_p[x_i]$. For $p = 2$, the same formulas prove $\mathrm{Tr}_{\mathbb{U}_i/\mathbb{U}_1}(\gamma_i) = 1 + \gamma_1$ for $i \geqslant 2$ and

$$\mathrm{Tr}_{\mathbb{U}_i/\mathbb{U}_0}(\gamma_i) = \begin{cases} 1 + \gamma_0 & \text{if } d \text{ even,} \\ 1 & \text{if } d \text{ odd.} \end{cases}$$

In both cases $\mathrm{Tr}_{\mathbb{U}_i/\mathbb{F}_p}(\gamma_i) = 1$, proving the analogue of Lemma 2. Lemma 3 is shown the same way by observing that $\gamma_0 \in \mathbb{F}_p[\gamma_i]$, for any d.

Composition. We give next an algorithm for polynomial composition, to be used in the construction of the tower defined before. Given P and R in $\mathbb{F}_p[X]$, we want to compute $P(R)$. For the cost analysis, it will be useful later on to consider both the degree k and the number of terms ℓ of R.

Compose is a recursive process that cuts P into $c+1$ "slices" of degree less than p^n, recursively composes them with R, and concludes using Horner's scheme and the linearity of the p-power. At the leaves of the recursion tree, we use a naive algorithm of cost $O(\deg(P)^2 k\ell)$.

Compose

Input $P, R \in \mathbb{F}_p[X]$ and $c, n \in \mathbb{N}$.
Output $P(R)$.

1. let $n = \lfloor \log_p(\deg(P)) \rfloor$ and $c = \deg(P)$ div p^n
2. If $n = 0$, return NaiveCompose(P, R)
3. write $P = \sum_{i=0}^{c} P_i X^{ip^n}$, with $P_i \in \mathbb{F}_p[X], \deg P_i < p^n$
4. for $i \in [0, \ldots, c]$, let $Q_i =$ Compose(P_i, R)
5. let $Q = 0$
6. for $i \in [c, \ldots, 0]$, let $Q = QR(X^{p^n}) + Q_i$
7. return Q

THEOREM 4. *If R has degree k and ℓ non-zero coefficients and if $\deg(P) = s$, then* Compose(P, R) *outputs $P(R)$ in time $O(ps \log_p(s) k\ell)$.*

PROOF. Correctness is clear, since $R^{p^n} = R(X^{p^n})$. To analyze the cost, we let $\mathsf{C}(c, n)$ be the cost of Compose when $\deg(P) \leqslant (c+1)p^n$, with $c < p$. Then $\mathsf{C}(c, 0) \in O(c^2 k\ell)$. For $n > 0$, at each pass in the loop at step 6, $\deg(Q) < cp^n k$, so that the multiplication (using the naive algorithm) and addition take time $O(cp^n k\ell)$. Thus the time spent in the loop is $O(c^2 p^n k\ell)$, and the running time satisfies

$$\mathsf{C}(c, n) \leqslant (c+1)\mathsf{C}(p-1, n-1) + O(c^2 p^n k\ell).$$

Let then $\mathsf{C}'(n) = \mathsf{C}(p-1, n)$, so that we have

$$\mathsf{C}'(0) \in O(p^2 k\ell), \quad \mathsf{C}'(n) \leqslant p\mathsf{C}'(n-1) + O(p^{n+2} k\ell).$$

We deduce that $\mathsf{C}'(n) \in O(p^{n+2} nk\ell)$, and finally $\mathsf{C}(c, n) \in O(cp^{n+1} nk\ell + c^2 p^n k\ell)$. The values c, n computed at step 1 of the top-level call to Compose satisfy $cp^n \leqslant s$ and $n \leqslant \log_p(s)$; this gives our conclusion. □

A binary divide-and-conquer algorithm [11, Ex. 9.20] has cost $O(\mathsf{M}(sk)\log(s))$. Our algorithm has a slightly better dependency on s, but adds a polynomial cost in p and l. However, we have in mind cases with p small and $\ell = 2$, where the latter solution is advantageous.

Computing the minimal polynomials. Theorem 1 shows that we have defined a primitive tower. To be able to work with it, we explain now how to compute the minimal polynomial Q_i of x_i over $\mathbb{F}_p$. This is done by extending Cantor's construction [6], which had $\mathbb{U}_0 = \mathbb{F}_p$.

For $i = 0$, we are given $Q_0 \in \mathbb{F}_p[X_0]$ such that $\mathbb{U}_0 = \mathbb{F}_p[X_0]/Q_0(X_0)$, so there is nothing to do; we assume that $\mathrm{Tr}_{\mathbb{U}_0/\mathbb{F}_p}(x_0) \neq 0$ to meet the hypotheses of Theorem 1. Remark that if this trace was zero, assuming $\gcd(d, p) = 1$, we could replace Q_0 by $Q_0(X_0 - 1)$; this is done by taking $R = X_0 - 1$ in algorithm Compose, so by Theorem 4 the cost is $O(pd \log_p(d))$.

For $i = 1$, we know that $x_1^p - x_1 = x_0$, so x_1 is a root of $Q_0(X_1^p - X_1)$. Since $Q_0(X_1^p - X_1)$ is monic of degree pd, we deduce that $Q_1 = Q_0(X_1^p - X_1)$. To compute it, we use algorithm Compose with arguments Q_0 and $R = X_1^p - X_1$; the cost is $O(p^2 d \log_p(d))$ by Theorem 4. The same arguments hold for $i = 2$ when $p = 2$ and d is odd.

To deal with other indexes i, we follow Cantor's construction. Let $\Phi \in \mathbb{F}_p[X]$ be the reduction modulo p of the $(2p-1)$th cyclotomic polynomial. Cantor implicitly works modulo an irreducible factor of Φ. The following shows that we can avoid factorization, by working modulo Φ.

LEMMA 5. *Let $A = \mathbb{F}_p[X]/\Phi$ and let $x = X \bmod \Phi$. For $Q \in \mathbb{F}_p[Y]$, define $Q^\star = \prod_{i=0}^{2p-2} Q(x^i Y)$. Then $Q^\star$ is in $\mathbb{F}_p[Y]$ and there exists $q^\star \in \mathbb{F}_p[Y]$ such that $Q^\star = q^\star(Y^{2p-1})$.*

PROOF. Let $F_1, \ldots, F_e$ be the irreducible factors of Φ and let f be their common degree. To prove that $Q^\star$ is in $\mathbb{F}_p[Y]$, we prove that for $j \leqslant e$, $Q_j^\star = Q^\star \bmod F_j$ is in $\mathbb{F}_p[Y]$ and independent from j; the claim follows by Chinese Remaindering.

For $j \leqslant e$, let a_j be a root of F_j in the algebraic closure of $\mathbb{F}_p$, so that $Q_j^\star = \prod_{i=0}^{2p-2} Q(a_j^i Y)$. Since $\gcd(p^f, 2p-1) = 1$, $Q_j^\star$ is invariant under $\mathrm{Gal}(\mathbb{F}_{p^f}/\mathbb{F}_p)$, and thus in $\mathbb{F}_p[Y]$. Besides, for $j, j' \leqslant e$, $a_j = a_{j'}^k$, for some k coprime to $2p-1$, so that $Q_j^\star = Q_{j'}^\star$, as needed.

To conclude, note that for $j \leqslant e$, $Q_j^\star(a_j Y) = Q_j^\star(Y)$, so that all coefficients of degree not a multiple of $2p-1$ are zero. Thus, $Q_j^\star$ has the form $q_j^\star(Y^{2p-1})$; by Chinese Remaindering, this proves the existence of the polynomial $q^\star$. □

We conclude as in [6]: supposing that we know the minimal polynomial Q_i of x_i over $\mathbb{F}_p$, we compute Q_{i+1} as follows. Since x_i is a root of Q_i, it is a root of $Q_i^\star$, so $\gamma_i = x_i^{2p-1}$ is a root of $q_i^\star$ and x_{i+1} is a root of $q_i^\star(Y^p - Y)$. Since the latter polynomial is monic of degree $p^{i+1}d$, it is the minimal polynomial Q_{i+1} of x_{i+1} over $\mathbb{F}_p$.

THEOREM 6. *Given Q_i, one can compute Q_{i+1} in time $O(p^{i+2}d\log_p(p^id) + \mathsf{M}(p^{i+2}d)\log(p))$.*

PROOF. Let $A = \mathbb{F}_p[X]/\Phi$. The algorithm of [3] computes Φ in time $O(p^2)$; then, polynomial multiplications in degree s in $A[Y]$ can be done in time $O(\mathsf{M}(sp))$ by Kronecker substitution. The overall cost of computing $Q_i^\star$ is $O(\mathsf{M}(p^{i+2}d)\log p)$ using [11, Algo. 10.3]. To get Q_{i+1} we use algorithm Compose with $R = Y^p - Y$, which costs $O(p^{i+2}d\log_p(p^id))$. □

The former cost is linear in $p^{i+2}d$, up to logarithmic factors, for an input of size p^id and an output of size $p^{i+1}d$.

Some further operations will be performed when we construct the tower: we will precompute quantities that will be of use in the algorithms of the next sections. Details are given in the next sections, when needed.

4. LEVEL EMBEDDING

We discuss here change-of-basis algorithms for the tower $(\mathbb{U}_0, \ldots, \mathbb{U}_k)$ of the previous section; these algorithms are needed for most further operations. We detail the main case where $P_i = X_i^p - X_i - X_{i-1}^{2p-1}$; the case $P_1 = X_1^p - X_1 - X_0$ (and $P_2 = X_2^2 + X_2 + X_1$ for $p = 2$ and d odd) is easier.

By Theorem 1, $\mathbb{U}_i$ equals $\mathbb{F}_p[X_{i-1}, X_i]/I$, where the ideal I admits the following Gröbner bases, for respectively the lexicographic orders $X_i > X_{i-1}$ and $X_{i-1} > X_i$:

$$\left|\begin{array}{r} X_i^p - X_i - X_{i-1}^{2p-1} \\ Q_{i-1}(X_{i-1}) \end{array}\right. \quad \text{and} \quad \left|\begin{array}{r} X_{i-1} - R_i(X_i) \\ Q_i(X_i), \end{array}\right.$$

with R_i in $\mathbb{F}_p[X_i]$. Since $\deg(Q_{i-1}) = p^{i-1}d$ and $\deg(Q_i) = p^id$, we associate the following $\mathbb{F}_p$-bases of $\mathbb{U}_i$ to each system:

$$\begin{aligned} \mathbf{D}_i &= (x_i^j,\ x_{i-1}x_i^j,\ \ldots,\ x_{i-1}^{p^{i-1}d-1}x_i^j)_{0\leqslant j<p}, \\ \mathbf{C}_i &= (1,\ x_i,\ \ldots,\ x_i^{p^id-1}). \end{aligned} \tag{2}$$

We describe an algorithm called Push-down which takes v written on the basis $\mathbf{C}_i$ and returns its coordinates on the basis $\mathbf{D}_i$; we also describe the inverse operation, called Lift-up. In other words, Push-down inputs $v \dashv \mathbb{U}_i$ and outputs the representation of v as

$$v = v_0 + v_1x_i + \cdots + v_{p-1}x_i^{p-1}, \quad \text{with all } v_j \dashv \mathbb{U}_{i-1} \tag{3}$$

and Lift-up does the opposite.

Hereafter, we let $\mathsf{L} : \mathbb{N} - \{0\} \to \mathbb{N}$ be such that both Push-down and Lift-up can be performed in time $\mathsf{L}(i)$; to simplify some expressions appearing later on, we add the mild constraints that $p\,\mathsf{L}(i) \leqslant \mathsf{L}(i+1)$ and $p\,\mathsf{M}(p^id) \in O(\mathsf{L}(i))$. To reflect the implementation's behavior, we also allow precomputations. These precomputations are performed when we build the tower; further details are at the end of this section.

THEOREM 7. *One can take $\mathsf{L}(i)$ in $O(p^{i+1}d\log_p(p^id)^2 + p\,\mathsf{M}(p^id))$.*

Remark that the input and output have size p^id; using fast multiplication, the cost is linear in $p^{i+1}d$, up to logarithmic factors. The rest of this section is devoted to proving this theorem. Push-down is a divide-and-conquer process, adapted to the shape of our tower; Lift-up uses classical ideas of trace computations (as in 2.3); the values we need will be obtained using the transposed version Push-down.

As said before, the algorithms of this section (and of the following ones) use precomputed quantities. To keep the pseudo-code simple, we do not explicitly list them in the inputs of the algorithms; we show, later, that the precomputation is fast too.

4.1 Modular multiplication

We first discuss a routine for multiplication by $X_i^{p^n}$ in $\mathbb{F}_p[Y, X_i]/(X_i^p - X_i - Y)$, and its transpose. We start by remarking that $X_i^{p^n} = X_i + R_n \bmod X_i^p - X_i - Y$, with

$$R_n = \textstyle\sum_{j=0}^{n-1} Y^{p^j}. \tag{4}$$

Then, precisely, for k in $\mathbb{N}$, we are interested in the operation $\mathsf{MulMod}_{k,n} : A \mapsto (X_i + R_n)A \bmod X_i^p - X_i - Y$, with $A \in \mathbb{F}_p[Y, X_i]$, $\deg(A, Y) < k$ and $\deg(A, X_i) < p$.

Since R_n is sparse, it is advantageous to use the naive algorithm; besides, to make transposition easy, we explicitly give the matrix of $\mathsf{MulMod}_{k,n}$. Let m_0 be the $(k + p^{n-1}) \times k$ matrix having 1's on the diagonal only, and for $\ell \leqslant p^{n-1}$, let m_ℓ be the matrix obtained from m_0 by shifting the diagonal down by ℓ places. Let finally m' be the sum $\Sigma_{j=0}^{n-1} m_{p^j}$. Then one verifies that the matrix of $\mathsf{MulMod}_{k,n}$ is

$$\begin{bmatrix} m' & & & & m_1 \\ m_0 & m' & & & m_0 \\ & m_0 & m' & & \\ & & \ddots & \ddots & \\ & & & m_0 & m' \end{bmatrix},$$

with columns indexed by $(X_i^j, \ldots, Y^{k-1}X_i^j)_{j<p}$ and rows by $(X_i^j, \ldots, Y^{k+p^{n-1}-1}X_i^j)_{j<p}$. Since this matrix has $O(pnk)$ non-zero entries, we can compute both $\mathsf{MulMod}_{k,n}$ and its dual $\mathsf{MulMod}^*_{k,n}$ in time $O(pnk)$.

4.2 Push-down

The input of Push-down is $v \dashv \mathbb{U}_i$, that is, given on the basis $\mathbf{C}_i$; we see it as a polynomial $V \in \mathbb{F}_p[X_i]$ of degree less than p^id. The output is the normal form of V modulo $X_i^p - X_i - X_{i-1}^{2p-1}$ and $Q_{i-1}(X_{i-1})$. We first use a divide-and-conquer subroutine to reduce V modulo $X_i^p - X_i - X_{i-1}^{2p-1}$; then, the result is reduced modulo $Q_{i-1}(X_{i-1})$ coefficient-wise.

To reduce V modulo $X_i^p - X_i - X_{i-1}^{2p-1}$, we first compute $W = V \bmod X_i^p - X_i - Y$, then we replace Y by X_{i-1}^{2p-1} in W. Because our algorithm will be recursive, we let $\deg(V)$ be arbitrary; then, we have the following estimate for W.

LEMMA 8. *We have $\deg(W, Y) \leqslant \deg(V)/p$.*

PROOF. Consider the matrix M of multiplication by X_i^p modulo $X_i^p - X_i - Y$; it has entries in $\mathbb{F}_p[Y]$. Due to the sparseness of the modulus, one sees that M has degree at most 1, and so M^k has coefficients of degree at most k. Thus, the remainders of $X_i^{pk}, \ldots, X_i^{pk+p-1}$ modulo $X_i^p - X_i - Y$ have degree at most k in Y. □

We compute W by a recursive subroutine Push-down-rec, similar to Compose. As before, we let c, n be such that $1 \leqslant c < p$ and $\deg(V) < (c+1)p^n$, so that we have

$$V = V_0 + V_1X_i^{p^n} + \cdots + V_cX_i^{cp^n},$$

with all V_j in $\mathbb{F}_p[X_i]$ of degree less than p^n. First, we recursively reduce $V_0, \ldots, V_c$ modulo $X_i^p - X_i - Y$, to obtain bivariate polynomials $W_0, \ldots, W_c$. Let R_n be the polynomial defined in Equation (4). Then, we get W by computing $\Sigma_{j=0}^{c} W_j(X_i + R_n)^j$ modulo $X_i^p - X_i - Y$, using Horner's scheme as in Compose. Multiplications by $X_i + R_n$ modulo $X_i^p - X_i - Y$ are done using MulMod.

Push-down-rec

Input $V \in \mathbb{F}_p[X_i]$ and $c, n \in \mathbb{N}$.
Output $W \in \mathbb{F}_p[Y, X_i]$.

1. if $n = 0$ return V
2. write $V = \sum_{j=0}^{c} V_j X_i^{jp^n}$, with $V_j \in \mathbb{F}_p[X_i], \deg V_j < p^n$
3. for $j \in [0, \dots, c]$, let $W_j =$ Push-down-rec$(V_j, p-1, n-1)$
4. $W = 0$
5. for $j \in [c, \dots, 0]$, let $W = \mathsf{MulMod}_{(c+1)p^{n-1},n}(W) + W_j$
6. return W

Push-down

Input $v \dashv \mathbb{U}_i$.
Output v written as $v_0 + \cdots + v_{p-1} x_i^{p-1}$ with $v_j \dashv \mathbb{U}_{i-1}$.

1. let V be the canonical preimage of v in $\mathbb{F}_p[X_i]$
2. let $n = \lfloor \log_p(p^i d - 1) \rfloor$ and $c = (p^i d - 1)$ div p^n
3. let $W =$ Push-down-rec(V, c, n)
4. let $Z =$ Evaluate$(W, [X_{i-1}^{2p-1}, X_i])$
5. let $Z = Z$ mod Q_{i-1}
6. return the residue class of Z mod $(X_i^p - X_i - X_{i-1}^{2p-1}, Q_{i-1})$

PROPOSITION 9. *Algorithm* Push-down *is correct and takes time* $O(p^{i+1} d \log_p(p^i d)^2 + p\,\mathsf{M}(p^i d))$.

PROOF. Correctness is straightforward; note that at step 5 of Push-down-rec, $\deg(W, Y) < (c+1)p^{n-1}$, so our call to $\mathsf{MulMod}_{(c+1)p^{n-1},n}$ is justified. By the claim of Subsection 4.1 on the cost of MulMod, the total time spent in that loop is $O(nc^2 p^n)$. As in Theorem 4, we deduce that the time spent in Push-down-rec is $O(n^2 c^2 p^n)$.

In Push-down, we have $cp^n < p^i d$ and $n < \log_p(p^i d)$, so the previous cost is $O(p^{i+1} d \log_p(p^i d)^2)$. Reducing one coefficient of Z modulo Q_{i-1} takes time $O(\mathsf{M}(p^i d))$, so step 5 has cost $O(p\,\mathsf{M}(p^i d))$. Step 6 is free, since at this stage Z is already reduced. □

4.3 Transposed push-down

We discuss here the transpose of Push-down. Push-down is the $\mathbb{F}_p$-linear change-of-basis from the basis $\mathbf{C}_i$ to $\mathbf{D}_i$, so its transpose takes an $\mathbb{F}_p$-linear form $\ell \in \mathbb{U}_i^*$ given by its values on $\mathbf{D}_i$, and outputs its values on $\mathbf{C}_i$. The input is the (finite) generating series $L = \Sigma_{a<p^{i-1}d,\, b<p} \ell(x_{i-1}^a x_i^b) X_{i-1}^a X_i^b$; the output is $M = \Sigma_{a<p^i d} \ell(x_i^a) X_i^a$.

As in [1], the transposed algorithm is obtained by reversing the initial algorithm step by step, and replacing subroutines by their transposes. The overall cost remains the same; we review here the main transformations.

In Push-down-rec, the initial loop at step 5 is a Horner scheme; the transposed loop is run backward, and its core becomes $L_j = L \bmod Y^{n-1}$ and $L = \mathsf{MulMod}^*_{(c+1)p^{n-1},n}(L)$; a small simplification yields the pseudo-code we give. In Push-down, after calling Push-down-rec, we evaluate W at $[X_{i-1}^{2p-1}, X_i]$: the transposed operation Evaluate* maps the series $\Sigma_{a,b} \ell_{a,b} X_{i-1}^a X_i^b$ to $\Sigma_{a,b} \ell_{(2p-1)a,b} Y^a X_i^b$. Then, originally, we perform a Euclidean division by Q_{i-1} on Z: the transposed algorithm mod* is in [1, Sect. 5.2].

Push-down-rec*

Input $L \in \mathbb{F}_p[Y, X_i]$ and $c, n \in \mathbb{N}$.
Output $M \in \mathbb{F}_p[X_i]$

1. If $n = 0$ return L
2. for $j \in [c, \dots, 0]$,
   - let $L_j = L \bmod Y^{n-1}$
   - let $M_j =$ Push-down-rec$^*(L_j, p-1, n-1)$
   - let $L = \mathsf{MulMod}^*_{(c+1)p^{n-1},n}(L)$
3. return $\sum_{j=0}^{c} M_j X_i^{jp^n}$

Push-down*

Input $L \in \mathbb{F}_p[X_{i-1}, X_i]$
Output $M \in \mathbb{F}_p[X_i]$

1. let $n = \lfloor \log_p(p^i d - 1) \rfloor$ and $c = (p^i d - 1)$ div p^n
2. let $P =$ mod$^*(L, Q_{i-1})$
3. let $M =$ Evaluate$^*(P, [X_{i-1}^{2p-1}, X_i])$
4. return Push-down-rec$^*(M, c, n)$

Lift-up

Input v written as $v_0 + \cdots + v_{p-1} x_i^{p-1}$ with $v_j \dashv \mathbb{U}_{i-1}$.
Output $v \dashv \mathbb{U}_i$.

1. let W be the canonical preimage of v in $\mathbb{F}_p[X_{i-1}, X_i]$
2. let $L =$ TransposedMul(W, S_i)
3. let $M =$ Push-down$^*(L)$
4. let $N = M \operatorname{rev}_{p^i d}(Q_i) \bmod X_i^{p^i d}$
5. let $V = \operatorname{rev}_{p^i d - 1}(N) {Q'_i}^{-1} \bmod Q_i$
6. return the residue class of V modulo Q_i

4.4 Lift-up

Let v be given on the basis $\mathbf{D}_i$ and let W be its canonical preimage in $\mathbb{F}_p[X_{i-1}, X_i]$. The lift-up algorithm finds V in $\mathbb{F}_p[X_i]$ such that $W = V \bmod (X_i^p - X_i - X_{i-1}^{2p-1}, Q_{i-1})$ and outputs the residue class of V modulo Q_i. Hereafter, we assume that both ${Q'_i}^{-1} \bmod Q_i$ and

$$S_i = \textstyle\sum_{a<p^{i-1}d,\, b<p} \mathrm{Tr}_{\mathbb{U}_i/\mathbb{F}_p}(x_{i-1}^a x_i^b) X_{i-1}^a X_i^b$$

are known (see the discussion below). Then, the algorithm implements the trace formulas given in Subsection 2.3

PROPOSITION 10. *Algorithm* Lift-up *is correct and takes time* $O(p^{i+1} d \log_p(p^i d)^2 + p\,\mathsf{M}(p^i d))$.

PROOF. As said in Subsection 2.3, the transposed multiplication of W with S_i gives values of $\ell = v \cdot \mathrm{Tr}_{\mathbb{U}_i/\mathbb{F}_p}$ by means of $L = \Sigma_{a<p^{i-1}d,\, b<p} \ell(x_{i-1}^a x_i^b) X_{i-1}^a X_i^b$. This is written TransposedMul in the pseudo-code; an algorithm of cost $O(\mathsf{M}(p^i d))$ for this is in [21, Coro. 2]. The last subsection showed that step 3 gives $M = \Sigma_{a<p^i d} \ell(x_i^a) X_i^a$. Then, correctness follows from Equations (1); the costs of steps 4 and 5 are $O(\mathsf{M}(p^i d))$ and step 6 is free since V is reduced. □

Propositions 9 and 10 prove Theorem 7. The precomputations, that are done at the construction of $\mathbb{U}_i$, are as follows. First, we need the values of the trace on the basis $\mathbf{D}_i$; they are obtained in time $O(\mathsf{M}(p^i d))$ by [21, Prop. 8]. Then, we need ${Q'_i}^{-1} \bmod Q_i$; this takes time $O(\mathsf{M}(p^i d) \log(p^i d))$ by fast extended GCD computation. These precomputations save logarithmic factors at best, but are useful in practice.

5. FROBENIUS AND PSEUDOTRACE

In this section, we describe algorithms computing Frobenius and pseudotrace operators, specific to the tower of Section 3; they are the keys to the algorithms of the next section.

The algorithms in this section and the next one closely follow Couveignes' [8]. However, the latter assumed the existence of a quasi-linear time algorithm for multiplication in some specific towers in the multivariate basis $\mathbf{B}_i$ of Subsection 2.1. To our knowledge, no such algorithm exists. We use here the univariate basis $\mathbf{C}_i$ introduced previously, which makes multiplication straightforward. However, several push-down and lift-up operations are now required to accommodate the recursive nature of the algorithm.

Our main purpose here is to compute the pseudotrace $\mathrm{T}_n : x \mapsto \sum_{\ell=0}^{n-1} x^{p^\ell}$, for n of the form $p^j d$. First, however,

we describe how to compute values of the iterated Frobenius operator $x \mapsto x^{p^{p^j d}}$. Any $v \in \mathbb{U}_j$ is left invariant by this latter map. For $j < i$, we get, similarly to (4):

$$x_i^{p^{p^j d}} = x_i + \beta_{i-1,j}, \quad \text{with} \quad \beta_{i-1,j} = \mathrm{T}_{p^j d}(\gamma_{i-1}). \tag{5}$$

The Frobenius algorithm follows: starting from $v \dashv \mathbb{U}_i$, we first write $v = v_0 + \cdots + v_{p-1}x_i^{p-1}$, with $v_h \dashv \mathbb{U}_{i-1}$; by (5) and the linearity of the Frobenius, we deduce that

$$v^{p^{p^j d}} = \textstyle\sum_{h=0}^{p-1} v_h^{p^{p^j d}} \left(x_i + \beta_{i-1,j}\right)^h.$$

Then, we compute all $v_h^{p^{p^j d}}$ recursively; the final sum is computed using Horner's scheme. This algorithm requires the values $\beta_{i',j}$ for $i' < i$: we suppose that they are precomputed (the discussion of how we precompute them follows). To analyze costs, we use the function L of Section 4.

IterFrobenius

Input v, i, j with $v \dashv \mathbb{U}_i$ and $j \geqslant 0$.
Output $v^{p^{p^j d}} \dashv \mathbb{U}_i$.
1. if $i \leqslant j$, return v
2. let $v_0 + v_1 x_i + \cdots + v_{p-1}x_i^{p-1} = \mathsf{Push\text{-}down}(v)$
3. for $h \in [0, \ldots, p-1]$, let $t_h = \mathsf{IterFrobenius}(v_h, i-1, j)$
4. let $F = 0$
5. for $h \in [p-1, \ldots, 0]$, let $F = t_h + (x_i + \beta_{i-1,j})F$
6. return $\mathsf{Lift\text{-}up}(F)$

THEOREM 11. *Algorithm* IterFrobenius *is correct and takes time* $O(i\,\mathsf{L}(i))$.

PROOF. Correctness is clear. We note $\mathsf{F}(i,j)$ for the cost for $v \in \mathbb{U}_i$, so that $\mathsf{F}(0,j) = \cdots = \mathsf{F}(j,j) = 0$. Each pass through step 5 involves a multiplication by $x_i + \beta_{i-1,j}$, of cost of $O(p\mathsf{M}(p^{i-1}d))$, assuming $\beta_{i-1,j} \dashv \mathbb{U}_{i-1}$ is known. Altogether, we deduce the recurrence relation

$$\mathsf{F}(i,j) \leqslant p\,\mathsf{F}(i-1,j) + 2\,\mathsf{L}(i) + O(p^2\mathsf{M}(p^{i-1}d)),$$

so $\mathsf{F}(i,j) \leqslant p\,\mathsf{F}(i-1,j) + O(\mathsf{L}(i))$, by assumptions on M and L. The conclusion follows, again by assumptions on L. □

Next, we compute pseudotraces. Given $v \dashv \mathbb{U}_i$, the naive algorithm doing repeated squaring takes time $O(n\mathsf{M}(p^i d)\log p)$ for computing $\mathrm{T}_n(v)$. In particular, with $n = d$, we use a function NaivePseudotrace with that cost in our pseudo-code. For higher values of n of the form $p^j d$, we use the following relation, whose verification is straightforward:

$$\mathrm{T}_{p^j d}(v) = \textstyle\sum_{\ell=0}^{p-1} \mathrm{T}_{p^{j-1}d}(v)^{p^{p^{j-1}d\ell}}.$$

Pseudotrace

Input v, i, j with $v \dashv \mathbb{U}_i$.
Output $T_{p^j d}(v) \dashv \mathbb{U}_i$.
1. if $j = 0$ return $\mathsf{NaivePseudotrace}(v, d)$
2. $t_0 = \mathsf{Pseudotrace}(v, i, j-1)$
3. for $h \in [1, \ldots, p-1]$, let $t_h = \mathsf{IterFrobenius}(t_{h-1}, i, j-1)$
4. return $t_0 + t_1 + \cdots + t_{p-1}$

THEOREM 12. *Algorithm* Pseudotrace *is correct and takes time* $\mathsf{PT}(i) = O(pi^2\mathsf{L}(i) + d\mathsf{M}(p^i d)\log p)$ *for* $j \leqslant i$.

PROOF. Correctness is clear. For the cost analysis, we write $\mathsf{PT}(i,j)$ for the cost on input i and j, so the naive algorithm gives $\mathsf{PT}(i,0) = O(d\mathsf{M}(p^i d)\log p)$. For $j > 0$, step 2 costs $\mathsf{PT}(i,j-1)$, step 3 costs $O(pi\mathsf{L}(i))$ by Theorem 11 and step 4 costs $O(p^{i+1}d)$. This gives $\mathsf{PT}(i,j) = \mathsf{PT}(i,j-1) + O(pi\mathsf{L}(i))$, and thus $\mathsf{PT}(i,j) \in O(pij\mathsf{L}(i)) + \mathsf{PT}(i,0)$. □

The cost is thus $O(p^{i+2}d + p^i d^2)$, up to logarithmic factors, for an input and output size of $p^i d$. Better could be done with respect to d, using fast modular composition algorithms in the NaivePseudotrace algorithm, as in [13].

Finally, we discuss precomputations. When we construct $\mathbb{U}_{i+1}$, we compute all $\beta_{i,j} = \mathrm{T}_{p^j d}(\gamma_i) \dashv \mathbb{U}_i$, for $j \leqslant i$, using the Pseudotrace algorithm. The inner calls to IterFrobenius only use pseudotraces that are already known. Besides, a single call to $\mathsf{Pseudotrace}(\gamma_i, i, i)$ actually computes *all* $\mathrm{T}_{p^j d}(\gamma_i)$ for $j \leqslant i$, in time $O(pi^2\mathsf{L}(i) + d\mathsf{M}(p^i d)\log p)$.

6. ARBITRARY TOWERS

Finally, we bring our previous algorithms to an arbitrary tower, using Couveignes' isomorphism algorithm [8]. As in the previous section, we adapt this algorithm to our context, by adding suitable push-down and lift-up operations.

Let Q_0 be irreducible of degree d in $\mathbb{F}_p[X_0]$, such that $\mathrm{Tr}_{\mathbb{U}_0/\mathbb{F}_p}(x_0) \neq 0$, with as before $\mathbb{U}_0 = \mathbb{F}_p[X_0]/Q_0$. We let $(G_i)_{0 \leqslant i < k}$ and $(\mathbb{U}_0, \ldots, \mathbb{U}_k)$ be as in Section 3.

We also consider another sequence $(G'_i)_{0 \leqslant i < k}$, that defines another tower $(\mathbb{U}'_0, \ldots, \mathbb{U}'_k)$. Since $(\mathbb{U}'_0, \ldots, \mathbb{U}'_k)$ is not necessarily primitive, we fall back to the multivariate basis of Subsection 2.1: we write elements of $\mathbb{U}'_i$ on the basis $\mathbf{B}'_i = \{x_0'^{e_0} \cdots x_i'^{e_i}\}$, with $x_0 = x'_0$, $0 \leqslant e_0 < d$ and $0 \leqslant e_j < p$ for $1 \leqslant j \leqslant i$.

To compute in $\mathbb{U}'_i$, we will use an isomorphism $\mathbb{U}'_i \to \mathbb{U}_i$. Such an isomorphism is determined by the images $\mathbf{s}_i = (s_0, \ldots, s_i)$ of $(x'_0, \ldots, x'_i)$, with $s_i \dashv \mathbb{U}_i$ (we always take $s_0 = x_0$). This isomorphism, denoted by $\sigma_{\mathbf{s}_i}$, takes as input v written on the basis $\mathbf{B}'_i$ and outputs $\sigma_{\mathbf{s}_i}(v) \dashv \mathbb{U}_i$.

To analyze costs, we use the functions L and PT introduced in the previous sections. We also let $2 \leqslant \omega \leqslant 3$ be a feasible exponent for linear algebra over $\mathbb{F}_p$ [11, Ch. 12].

THEOREM 13. *Given* Q_0 *and* $(G'_i)_{0 \leqslant i < k}$, *one can find* $\mathbf{s}_k = (s_0, \ldots, s_k)$ *in time* $O(d^\omega k + \mathsf{PT}(k) + \mathsf{M}(p^{k+1}d)\log(p))$. *Once they are known, one can apply* $\sigma_{\mathbf{s}_k}$ *and* $\sigma_{\mathbf{s}_k}^{-1}$ *in time* $O(k\,\mathsf{L}(k))$.

Thus, we can compute products, inverses, etc, in $\mathbb{U}'_k$ for the cost of the corresponding operation in $\mathbb{U}_k$, plus $O(k\,\mathsf{L}(k))$.

6.1 Solving Artin-Schreier equations

As a preliminary, given $\alpha \dashv \mathbb{U}_i$, we discuss how to solve the Artin-Schreier equation $X^p - X = \alpha$ in $\mathbb{U}_i$. We assume that $\mathrm{Tr}_{\mathbb{U}_i/\mathbb{F}_p}(\alpha) = 0$, so this equation has solutions in $\mathbb{U}_i$.

Because $X^p - X$ is $\mathbb{F}_p$-linear, the equation can be directly solved by linear algebra, but this is too costly. In [8], Couveignes gives a solution adapted to our setting, that reduces the problem to solving Artin-Schreier equations in $\mathbb{U}_0$. Given a solution $\delta \in \mathbb{U}_i$ of the equation $X^p - X = \alpha$, he observes that any solution μ of

$$X^{p^{p^{i-1}d}} - X = \eta, \quad \text{with} \quad \eta = \mathrm{T}_{p^{i-1}d}(\alpha). \tag{6}$$

is of the form $\mu = \delta - \Delta$ with $\Delta \in \mathbb{U}_{i-1}$, hence Δ is a root of

$$X^p - X - \alpha + \mu^p - \mu. \tag{7}$$

This equation has solutions in $\mathbb{U}_{i-1}$ by hypothesis and hence it can be solved recursively. First, however, we tackle the problem of finding a solution of (6).

For this purpose, observe that the left hand side of (6) is

$\mathbb{U}_{i-1}$-linear and its matrix on the basis $(1, \ldots, x_i^{p-1})$ is

$$\begin{bmatrix} 0 & \binom{1}{0}\beta_{i-1,i-1} & \cdots & \binom{p-1}{0}\beta_{i-1,i-1}^{p-1} \\ & \ddots & & \vdots \\ & & 0 & \binom{p-1}{p-2}\beta_{i-1,i-1} \\ & & & 0 \end{bmatrix}$$

Then, algorithm ApproximateAS finds the required solution.

ApproximateAS

Input $\eta \dashv \mathbb{U}_i$ such that (6) has a solution.
Output $\mu \dashv \mathbb{U}_i$ solution of (6).
1. let $\eta_0 + \eta_1 x_i + \cdots + \eta_{p-2} x_i^{p-2} = \mathsf{Push\text{-}down}(\eta)$
2. for $j \in [p-1, \ldots, 1]$,
   let $\mu_j = \frac{1}{j\tau}\left(\eta_{j-1} - \sum_{h=j+1}^{p-1} \binom{h}{j-1}\beta_{i-1,i-1}^{h-j+1}\mu_h\right)$
3. return $\mathsf{Lift\text{-}up}(\mu_1 x_i + \ldots + \mu_{p-1}x_i^{p-1})$

THEOREM 14. *Algorithm* ApproximateAS *is correct and takes time* $O(\mathsf{L}(i))$.

PROOF. Correctness is clear from Gaussian elimination. For the cost analysis, remark that $\beta_{i-1,i-1}$ has already been precomputed to permit iterated Frobenius and pseudotrace computations. Step 2 takes $O(p^2)$ additions and scalar operations in $\mathbb{U}_{i-1}$; the overall cost is dominated by that of the push-down and lift-up by assumptions on L. □

Writing the recursive algorithm is now straightforward. To solve Artin-Schreier equations in $\mathbb{U}_0$, we use a naive algorithm based on linear algebra, written NaiveSolve.

Artin-Schreier

Input α, i such that $\alpha \dashv \mathbb{U}_i$ and $\mathrm{Tr}_{\mathbb{U}_i/\mathbb{F}_p}(\alpha) = 0$.
Output $\delta \dashv \mathbb{U}_i$ such that $\delta^p - \delta = \alpha$.
1. if $i = 0$, return $\mathsf{NaiveSolve}(X^p - X - \alpha)$
2. let $\eta = \mathsf{Pseudotrace}(\alpha, i, i-1)$
3. let $\mu = \mathsf{ApproximateAS}(\eta)$
4. let $\alpha_0 = \mathsf{Push\text{-}down}(\alpha - \mu^p + \mu)$
5. let $\Delta = \mathsf{Artin\text{-}Schreier}(\alpha_0, i-1)$
6. return $\mu + \mathsf{Lift\text{-}up}(\Delta)$

THEOREM 15. *Algorithm* Artin-Schreier *is correct and takes time* $O(d^\omega + \mathsf{PT}(i))$.

PROOF. Correctness follows from the previous discussion. For the complexity, note $\mathsf{AS}(i)$ the cost for $\alpha \dashv \mathbb{U}_i$. The cost $\mathsf{AS}(0)$ of the naive algorithm is $O(\mathsf{M}(d)\log(p) + d^\omega)$, where the first term is the cost of computing x_0^p and the second one the cost of linear algebra.

When $i \geqslant 1$, step 2 has cost $\mathsf{PT}(i)$, steps 3, 4 and 6 all contribute $O(\mathsf{L}(i))$ and step 5 contributes $\mathsf{AS}(i-1)$. The most important contribution is at step 2, hence $\mathsf{AS}(i) = \mathsf{AS}(i-1) + O(\mathsf{PT}(i))$. The assumptions on L imply that the sum $\mathsf{PT}(1) + \cdots + \mathsf{PT}(i)$ is $O(\mathsf{PT}(i))$. □

6.2 Applying the isomorphism

We get back to the isomorphism question. We assume that $\mathbf{s}_i = (s_0, \ldots, s_i)$ is known and we give the cost of applying $\sigma_{\mathbf{s}_i}$ and its inverse. We first discuss the forward direction.

As input, $v \in \mathbb{U}'_i$ is written on the multivariate basis $\mathbf{B}'_i$ of $\mathbb{U}'_i$; the output is $t = \sigma_{\mathbf{s}_i}(v) \dashv \mathbb{U}_i$. As before, the algorithm is recursive: we write $v = \Sigma_{j<p} v_j(x'_0, \ldots, x'_{i-1}){x'_i}^j$, whence

$$\sigma_{\mathbf{s}_i}(v) \;=\; \textstyle\sum_{j<p} \sigma_{\mathbf{s}_i}(v_j) s_i^j \;=\; \sum_{j<p} \sigma_{\mathbf{s}_{i-1}}(v_j) s_i^j \;;$$

the sum is computed by Horner's scheme. To speed-up the computation, it is better to perform the latter step in a bivariate basis, that is, through a push-down and a lift-up.

Given $t \dashv \mathbb{U}_i$, to compute $v = \sigma_{\mathbf{s}_i}^{-1}(t)$, we run the previous algorithm backward. We first push-down t, obtaining $t = t_0 + \cdots + t_{p-1}x_i^{p-1}$, with all $t_j \dashv \mathbb{U}_{i-1}$. Next, we rewrite this as $t = t'_0 + \cdots + t'_{p-1}s_i^{p-1}$, with all $t'_j \dashv \mathbb{U}_{i-1}$, and it suffices to apply $\sigma_{\mathbf{s}_i}^{-1}$ (or equivalently $\sigma_{\mathbf{s}_{i-1}}^{-1}$) to all t'_i. The non-trivial part is the computation of the t'_j: this is done by applying the algorithm FindParametrization mentioned in Subsection 2.3, in the extension $\mathbb{U}_i = \mathbb{U}_{i-1}[X_i]/P_i$.

ApplyIsomorphism

Input v, i with $v \in \mathbb{U}'_i$ written on the basis $\mathbf{B}'_i$.
Output $\sigma_{\mathbf{s}_i}(v) \dashv \mathbb{U}_i$.
1. if $i = 0$ then return v
2. write $v = \Sigma_{j<p} v_j(x'_0, \ldots, x'_{i-1}){x'_i}^j$
3. let $s_{i,0} + \cdots + s_{i,p-1}x_i^{p-1} = \mathsf{Push\text{-}down}(s_i)$
4. for $j \in [0, \ldots, p-1]$ let $t_j = \mathsf{ApplyIsomorphism}(v_j, i-1)$
5. let $t = 0$
6. for $j \in [p-1, \ldots, 0]$ let $t = (s_{i,0} + \cdots + s_{i,p-1}x_i^{p-1})t + t_j$
7. return $\mathsf{Lift\text{-}up}(t)$

ApplyInverse

Input t, i with $t \dashv \mathbb{U}_i$.
Output $\sigma_{\mathbf{s}_i}^{-1}(t) \in \mathbb{U}'_i$ written on the basis $\mathbf{B}'_i$.
1. if $i = 0$ then return t
2. let $t_0 + \cdots + t_{p-1}x_i^{p-1} = \mathsf{Push\text{-}down}(t)$
3. let $s_{i,0} + \cdots + s_{i,p-1}x_i^{p-1} = \mathsf{Push\text{-}down}(s_i)$
4. let $t'_0 + \cdots + t'_{p-1}X^{p-1} = \mathsf{FindParametrization}(t_0 + \cdots + t_{p-1}x_i^{p-1}, s_{i,0} + \cdots + s_{i,p-1}x_i^{p-1})$
5. return $\Sigma_{j<p}\mathsf{ApplyInverse}(t'_j, i-1){x'_i}^j$

PROPOSITION 16. *Algorithms* ApplyIsomorphism *and* ApplyInverse *are correct and both take time* $O(i\mathsf{L}(i))$.

PROOF. In both cases, correctness is clear, since the algorithms translate the former discussion. As to complexity, in both cases, we do p recursive calls, $O(1)$ push-downs and lift-ups, and a few extra operations: for ApplyIsomorphism, these are p multiplications / additions in the bivariate basis $\mathbf{D}_i$ of Section 4; for ApplyInverse, this is calling the algorithm FindParametrization of Subsection 2.3. The costs are $O(p\mathsf{M}(p^i d))$ and $O(p^2\mathsf{M}(p^{i-1}d))$, which are in $O(\mathsf{L}(i))$ by assumption on L. We conclude as in Theorem 11. □

6.3 Proof of Theorem 13

Finally, assuming that only $(s_0, \ldots, s_{i-1})$ are known, we describe how to determine s_i. Several choices are possible: the only constraint is that s_i should be a root of $X_i^p - X_i - \sigma_{\mathbf{s}_i}(\gamma'_{i-1}) = X_i^p - X_i - \sigma_{\mathbf{s}_{i-1}}(\gamma'_{i-1})$ in $\mathbb{U}_i$.

Using Proposition 16, we can compute $\alpha = \sigma_{\mathbf{s}_{i-1}}(\gamma'_{i-1}) \dashv \mathbb{U}_{i-1}$ in time $O((i-1)\mathsf{L}(i-1)) \subset O(i\mathsf{L}(i))$. Applying a lift-up to α, we are then in the conditions of Theorem 15, so we can find s_i for an extra $O(d^\omega + \mathsf{PT}(i))$ operations.

We can then summarize the cost of all precomputations: to the cost of determining $\mathbf{s}_i$, we add the costs related to the tower $(\mathbb{U}_0, \ldots, \mathbb{U}_i)$, given in Sections 3, 4 and 5. After a few simplifications, we obtain the upper bound $O(d^\omega + \mathsf{PT}(i) + \mathsf{M}(p^{i+1}d)\log(p))$. Summing over i gives the first claim of the theorem. The second is a restatement of Proposition 16.

7. EXPERIMENTAL RESULTS

We describe here the implementation of our algorithms and an application coming from elliptic curve cryptology.

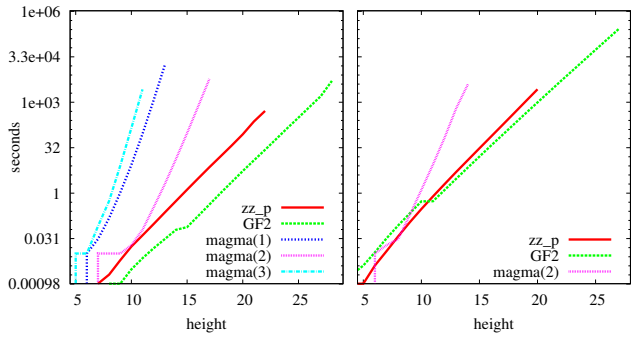
Experimental results. The previous algorithms are implemented on top the NTL C++ library [23] compiled with

the gf2x package [4], which provide the basic univariate polynomial arithmetic needed here. Our implementation handles three NTL classes of finite fields: `GF2` for $p = 2$, `zz_p` for word-size p and `ZZ_p` for arbitrary p.

We compare our timings with those obtained in Magma [2]. We take $p = 2$ and $d = 1$ (that is, $\mathbb{U}_0 = \mathbb{F}_p$); the x-coordinate gives the number of levels we construct and the y-coordinate gives timings in seconds, in *logarithmic* scale. All results are obtained on an AMD Opteron 250 (2.4GHz).

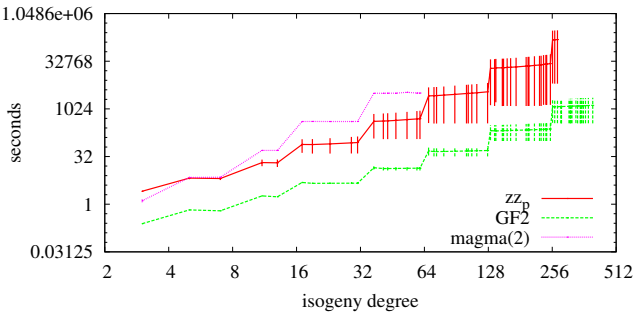
We have two ways of doing arithmetic modulo 2 in NTL: `GF2` is specialized to $p = 2$; `zz_p` is more general. In Magma, there exist several ways to build field extensions:

`quo<U|P>` builds the quotient of the univariate polynomial ring U by $P \in U$ (written magma(1) hereafter);
`ext<k|P>` builds the extension of the field k by $P \in k[X]$ (magma(2));
`ext<k|p>` builds an extension of degree p of k (magma(3)).



Our first graph gives timings for the construction of the tower of Section 3; the second one gives timings for constructing an isomorphism with an arbitrary tower (in Magma, only the magma(2) approach was meaningful). The timings of our code are significantly better.

Isogeny algorithm. An isogeny is a regular map between two elliptic curves $\mathscr{E}$ and $\mathscr{E}'$ that is also a group morphism. Our interest is Couveignes' isogeny algorithm [7], which computes isogenies of degree $\sim p^k$. Couveignes' later paper [8] described improvements to speed up the computation, but as we already mentioned, a key component, fast arithmetic in Artin-Schreier towers, was still missing. The original algorithm of [7] was first implemented in [16]; using this paper's algorithms, it now becomes possible to have a completely explicit version of the fast variant. The algorithm relies on the interpolation of a rational function at special points in an Artin-Schreier tower; the Master thesis [9] describes improved algorithms for this task, along the lines of [10]. Its running time is probabilistic; we plot the average running times with bars around them for minimum/maximum times; the distribution is uniform.



To highlight the benefits of this paper, we compare a Magma implementation to our C++ code, for the same variant of the isogeny algorithm, on an Intel Xeon E5430 (2.6GHz). For $p = 2$, it should be noted that Lercier's isogeny algorithm [15] has better performance; for generic, small, p we mention as well a new algorithm by Lercier and Sirvent [17] which still lacks an implementation.

Acknowledgments. We thank J.-M. Couveignes and F. Morain for useful discussions. We acknowledge financial support from the INRIA "Équipes associées" ECHECS team, NSERC and the Canada Research Chair program.

8. REFERENCES

[1] A. Bostan, G. Lecerf, and É. Schost. Tellegen's principle into practice. In *ISSAC'03*, pages 37–44. ACM, 2003.
[2] W. Bosma, J. Cannon, C. Playoust. The Magma algebra system. I. The user language. J. Symb. Comp., 24(3-4):235-265, 1997.
[3] R. P. Brent. On computing factors of cyclotomic polynomials. *Math. Comp.* 61:131–149, 1993.
[4] R. Brent, P. Gaudry, E. Thomé, P. Zimmermann. Faster multiplication in GF(2)[x]. In *ANTS'08*, 153-166. Springer, 2008.
[5] P. Bürgisser, M. Clausen, and A. Shokrollahi. *Algebraic complexity theory*. Springer–Verlag, 1997.
[6] D. G. Cantor. On arithmetical algorithms over finite fields. *Journal of Combinatorial Theory*, Series A 50, 285-300, 1989.
[7] J.-M. Couveignes. Computing ℓ-isogenies using the p-torsion. in *ANTS'II*, 59–65. Springer, 1996.
[8] J.-M. Couveignes. Isomorphisms between Artin-Schreier towers. *Math. Comp.* 69(232): 1625–1631, 2000.
[9] L. De Feo. Calculs d'isogénies. M. Sc. Thesis, École polytechnique, 2007, `http://www.lix.polytechnique.fr/~defeo/`
[10] A. Enge and F. Morain, Fast decomposition of polynomials with known Galois group. in *AAECC-15*, 254–264. Springer, 2003.
[11] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999.
[12] J. von zur Gathen and J. Gerhard, Arithmetic and factorization of polynomials over $\mathbb{F}_2$. In *ISSAC'96*, pages 1–9. ACM, 1996.
[13] J. von zur Gathen and V. Shoup. Computing Frobenius maps and factoring polynomials. *Comp. Complex.*, 2(3):187–224, 1992.
[14] E. Kaltofen. Challenges of symbolic computation: my favorite open problems. *J. Symb. Comp.*, 29(6):891–919, 2000.
[15] R. Lercier. Computing isogenies in GF(2^n). In *ANTS-II*, LNCS vol 1122, pages 197–212. Springer, 1996.
[16] R. Lercier. Algorithmique des courbes elliptiques dans les corps finis. Ph.D. Thesis, École polytechnique, 1997.
[17] R. Lercier, T. Sirvent. On Elkies subgroups of ℓ-torsion points in curves defined over a finite field. To appear in *J. Théor. Nombres Bordeaux*.
[18] X. Li, M. Moreno Maza, and É. Schost. Fast arithmetic for triangular sets: from theory to practice. In *ISSAC'07*, pages 269–276. ACM, 2007.
[19] R. Lidl and H. Niederreiter. *Finite Fields*, second edition. Cambridge University Press, 1997.
[20] T. Mateer. Fast Fourier transform algorithms with applications. Ph.D. Thesis, Clemson University, August 2008.
[21] C. Pascal and É. Schost. Change of order for bivariate triangular sets. In *ISSAC'06*, pages 277–284. ACM, 2006.
[22] F. Rouillier. Solving zero-dimensional systems through the Rational Univariate Representation. *Appl. Alg. in Eng. Comm. Comput.*, 9(5):433–461, 1999.
[23] V. Shoup. NTL: A library for doing number theory. `http://www.shoup.net/ntl/`.
[24] V. Shoup. Fast construction of irreducible polynomials over finite fields. *J. Symb. Comp.* 17:371-391, 1994.
[25] V. Shoup. Efficient computation of minimal polynomials in algebraic extensions of finite fields. In *ISSAC'99*, pages 53–58, ACM, 1999.
[26] Y. Wang and X. Zhu. A Fast Algorithm for Fourier Transform Over Finite Fields and its VLSI Implementation. IEEE Journal on Selected Areas in Communications, 6 (3):572-7, 1988.