Fast Arithmetics in Artin-Schreier Towers over Finite Fields

Luca De Feo

LIX, École Polytechnique, Palaiseau, France

IRMAR, Université de Rennes 1, Rennes, France

Éric Schost

ORCCA and CSD, The University of Western Ontario, London, ON

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.

Key words: 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 $1 \leq i \leq k$; 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

[⋆] This research was partly supported by the INRIA "Équipes associées" ECHECS team, NSERC and the Canada Research Chair program.

Email addresses: `luca.defeo@polytechnique.edu` (Luca De Feo), `eschost@uwo.ca` (Éric Schost).

instance, when computing p^k-torsion groups of Abelian varieties over $\mathbb{F}_p$). The need for fast arithmetic in these towers is motivated in particular by applications to isogeny computation and point-counting in cryptography, as in (Couveignes, 1996).

Our contribution. The purpose of this paper is to give fast algorithms for arithmetic operations in Artin-Schreier towers. Prior results for this task are due to Cantor (1989) and Couveignes (2000). However, the algorithms of Couveignes (2000) need as a prerequisite a fast multiplication algorithm in some towers of a special kind, called "Cantor towers" in (Couveignes, 2000). Such an algorithm is unfortunately not in the literature, making the results of Couveignes (2000) 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 (Couveignes, 2000). Along the way, we also extend constructions of Cantor to the case of a general finite base field $\mathbb{U}_0$, where Cantor had $\mathbb{U}_0 = \mathbb{F}_p$. We present our implementation, in a library called `FAAST`, based on the library `NTL` by Shoup (2003). As an application, we put to practice the isogeny computation algorithm by Couveignes (1996) (or, more precisely, its refined version by De Feo (2011)).

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 in $\mathbb{F}_p[X]$ of degree less than n can be multiplied in $\mathsf{M}(n)$ operations, under the conditions of (von zur Gathen and Gerhard, 1999, 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 (von zur Gathen and Gerhard, 1999, Ch. 9 & 11).

The cost of *modular composition*, that is, of computing $F(G) \bmod H$, for $F, G, H \in \mathbb{F}_p[X]$ of degrees at most n, will be written $\mathsf{C}(n)$. We refer to von zur Gathen and Gerhard (1999, Ch. 12) for a presentation of known results in an algebraic computational model: the best known algorithms have subquadratic (but superlinear) cost in n. Note that in a boolean RAM model, the algorithm by Kedlaya and Umans (2008) takes quasi-linear time.

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 by Cantor (1989) and Couveignes (2000, 1996); 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 (1988) and have been extended in another direction (fast polynomial multiplication over arbitrary finite fields) by von zur Gathen and Gerhard (2002) and Mateer (2008).

This paper is an expanded version of the conference paper (De Feo and Schost, 2009). We provide a more thorough description of the properties of Cantor towers (Section 3), improvements to some algorithms (e.g. the Frobenius or pseudo-trace computations) and a more extensive experimental section.

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 the algorithm for isomorphisms between Artin-Schreier towers by Couveignes (2000) in our context, which yields fast arithmetics for *any* Artin-Schreier tower. Finally, Section 7 presents our implementation of the `FAAST` library and gives experimental results obtained by applying our algorithms to the isogeny algorithm by Couveignes (1996) for elliptic curves.

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

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

in $\mathbb{F}_p[X_0,\dots,X_i]$, and if $\mathbb{U}_i$ is a field. The residue class of X_i (resp. G_{i-1}) in $\mathbb{U}_i$, and thus in $\mathbb{U}_{i+1},\dots$, is written x_i (resp. γ_{i-1}), 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 $(\mathbb{U}_0,\dots,\mathbb{U}_k)$ 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 $0\le e_0<d$ and $0\le e_j<p$ for $1\le j\le i$. However, in this basis, we do not have very efficient arithmetic operations, starting from multiplication. Indeed, the natural approach to multiplication in $\mathbf{B}_i$ consists in a polynomial multiplication, followed by reduction modulo $(Q_0,P_1,\dots,P_i)$; however, the initial product gives a polynomial of partial degrees $(2d-2,2p-2,\dots,2p-2)$, so the number of monomials appearing is not linear in $[\mathbb{U}_i:\mathbb{F}_p]=p^id$. See (Li, Moreno Maza, and Schost, 2007) 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 the minimal polynomial of x_i, of degree p^id. In a primitive tower, unless otherwise stated, we represent the elements of $\mathbb{U}_i$ in the $\mathbb{F}_p$-basis $\mathbf{C}_i=(x_i^a \mid 0\le a<p^id)$.

To stress the fact that $v\in\mathbb{U}_i$ is represented in the basis $\mathbf{C}_i$, we write $v\vdash\mathbb{U}_i$. In this basis, assuming Q_i is known, additions and subtractions are done in p^id operations, multiplications in $O(\mathsf{M}(p^i d))$ operations (von zur Gathen and Gerhard, 1999, Ch. 9) and inversions in $O(\mathsf{M}(p^i d)\log(p^i d))$ operations (von zur Gathen and Gerhard, 1999, Ch. 11).

Note that having fast arithmetic operations in $\mathbb{U}_i$ enables us to write fast algorithms for polynomial arithmetic in $\mathbb{U}_i[Y]$, where Y is a new variable. Extending the previous notation, let us write $A \vdash \mathbb{U}_i[Y]$ to indicate that a polynomial $A \in \mathbb{U}_i[Y]$ is written in the basis $(x_i^\alpha Y^\beta)_{0\le\alpha<p^i d, 0\le\beta}$ of $\mathbb{U}_i[Y]$. Then, given $A, B \vdash \mathbb{U}_i[Y]$, both of degrees less than n, one can compute $AB \vdash \mathbb{U}_i[Y]$ in $O(\mathsf{M}(p^i dn))$ operations using Kronecker's substitution (von zur Gathen and Shoup, 1992, Lemma 2.2).

One can extend the fast Euclidean division algorithm to this context, as Newton iteration reduces Euclidean division to polynomial multiplication. The analysis of (von zur Gathen and Gerhard, 1999, Ch. 9) implies that Euclidean division of a degree n polynomial $A \vdash \mathbb{U}_i[Y]$ by a monic degree m polynomial $B \vdash \mathbb{U}_i[Y]$, with $m \le n$, can be done in $O(\mathsf{M}(p^i dn))$ operations.

Finally, fast GCD techniques carry over as well, as they are based on multiplication and division. Using the analysis of (von zur Gathen and Gerhard, 1999, Ch. 11), we see that the extended GCD of two monic polynomials $A, B \vdash \mathbb{U}_i[Y]$ of degree at most n can be computed in $O(\mathsf{M}(p^i dn\log(n)))$ operations.

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 $\deg(Q) = n$. 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:

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

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

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

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

Following Couveignes (2000), we also use a generalization of the trace. The nth *pseudotrace* of order m is the $\mathbb{F}_{p^m}$-linear operator

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

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 $m = [\mathbb{U}_j : \mathbb{F}_p] = p^j d$, $\mathrm{T}_{(n,m)}(v)$ 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 in a field extension of $\mathbb{U}_i$, 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 the existence of an algorithm with the same complexity, up to $O(r+s)$, that performs the transposed product $c \mapsto M^t c$; see (Bürgisser, Clausen, and Shokrollahi, 1997; Kaltofen, 2000; Bostan, Lecerf, and Schost, 2003). However, making the transposed algorithm explicit is not always straightforward; we will devote part of Section 4 to this issue.

We give here first consequences of this principle, after (Bostan, Lecerf, and Schost, 2003; Shoup, 1999, 1994). Consider a degree n field extension $\mathbb{U} \to \mathbb{U}'$, where $\mathbb{U}'$ is seen as an $\mathbb{U}$-vector space. 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 any 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 using T operations in $\mathbb{U}$. Then by the transposition principle, given w in $\mathbf{D}$ and ℓ in the dual basis $\mathbf{D}^*$, we can compute $w \cdot \ell$ in the dual basis $\mathbf{D}^*$ using $T + O(n)$ operations in $\mathbb{U}$. This was discussed already by Shoup (1999) and Bostan, Lecerf, and Schost (2003), we will get back to this in Section 4.

Suppose finally that $\mathbb{U}'$ is separable over $\mathbb{U}$ and that $b \in \mathbb{U}'$ generates $\mathbb{U}'$ over $\mathbb{U}$; we will denote by $Q \in \mathbb{U}[X]$ the minimal polynomial of b. 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 $\mathrm{rev}_e(P) = X^e P(1/X) \in \mathbb{U}[X]$. Then, recalling that $n = [\mathbb{U}' : \mathbb{U}]$, we define $\ell = w \cdot \mathrm{Tr}_{\mathbb{U}'/\mathbb{U}} \in \mathbb{U}'^*$ and

$$M = \sum_{j<n} \ell(b^j) X^j, \quad N = M\,\mathrm{rev}_n(Q) \bmod X^n. \tag{1}$$

This construction solves our problem: Rouillier (1999, Theorem 3.1) shows that $w = A(b)$, with $A = \mathrm{rev}_{n-1}(N) {Q'}^{-1} \bmod Q$. We will hereafter denote by $\mathsf{FindParameterization}(b, w)$ a subroutine that computes this polynomial A; it follows closely a similar algorithm by Shoup (1994).

Since this is the case we will need later on, we give details for the case where $\mathbb{U}'$ is presented as $\mathbb{U}' = \mathbb{U}[X]/P$, with P an Artin-Schreier polynomial (so $n = p$), and where the minimal polynomial Q of b is Artin-Schreier too. We denote by x the residue class of X in $\mathbb{U}[X]/P$, then by $\mathbf{P}_3$

$$\mathrm{Tr}_{\mathbb{U}'/\mathbb{U}} = (0, \dots, 0, -1) \qquad \text{in the basis } (1, x, \dots, x^{p-1}),$$

and for some $\alpha \in \mathbb{U}$

$$Q = X^p - X - \alpha, \qquad \mathrm{rev}_p\, Q = 1 - X^{p-1} - \alpha X^p, \qquad Q' = -1.$$

Proposition 1. Under the assumptions above, the algorithm $\mathsf{FindParameterization}$ is correct and requires $p-1$ multiplications in the $\mathbb{U}$-basis $(1, x, \dots, x^{p-1})$ of $\mathbb{U}'$, plus $O(p^2)$ operations $(+, \times)$ in $\mathbb{U}$.

FindParameterization

```
Input $w \in \mathbb{U}'$ written as $w_0 + \cdots + w_{p-1}x^{p-1}$, $b \in \mathbb{U}'$ written as $b_0 + \cdots + b_{p-1}x^{p-1}$.
Output A polynomial $A$ of degree less than $p$ such that $w = A(b)$.
  (1) let $\ell = w \cdot \mathrm{Tr}_{\mathbb{U}'/\mathbb{U}} = -(0, \dots, 0, w_{p-1}) - (w_{p-1}, \dots, w_1, w_0)$;
  (2) let $M = \sum_{j<p} \ell(b^j) X^j$;
  (3) let $N = M(1 - X^{p-1})$ mod $X^p$;
  (4) return $-\mathrm{rev}_{p-1}(N)$.
```

Proof. Correctness follows from the discussion above; to verify that $w \cdot \mathrm{Tr}_{\mathbb{U}'/\mathbb{U}}$ is indeed equal to the $\mathbb{U}$-linear form given in step 1, simply write the multiplication matrix of w and multiply by the trace form $(0, \dots, 0, -1)$.

Then, step 1 requires one addition in $\mathbb{U}$. Step 2 requires $p-1$ multiplications in $\mathbb{U}'$, plus $O(p^2)$ operations in $\mathbb{U}$. Step 3 requires one addition in $\mathbb{U}$. Finally, step 4 simply amounts to read the polynomial from right to left and change signs. □

Note that this cost can be improved with respect to p, by using the transposition principle and fast modular composition as in (Shoup, 1994, 1999); we do not give details, as this would not improve the overall complexity of the algorithms of the next sections.

3. A primitive tower

Our first task in this section is to describe a specific Artin-Schreier tower where arithmetic operations will be fast; then, we explain how to construct this tower.

3.1. Definition

The following theorem extends results by Cantor (1989, Th. 1.2), who dealt with the case $\mathbb{U}_0 = \mathbb{F}_p$.

Theorem 2. 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 $(G_i)_{0 \le i < k}$ be defined by

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

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

As before, for $i \ge 1$, let $P_i = X_i^p - X_i - G_{i-1}$ and for $i \ge 0$, let K_i be the ideal $\langle Q_0, P_1, \dots, P_i \rangle$ in $\mathbb{F}_p[X_0, \dots, X_i]$. Then the theorem says that for $i \ge 0$, $\mathbb{U}_i = \mathbb{F}_p[X_0, \dots, X_i]/K_i$ is a field, and that $x_i = X_i \bmod K_i$ generates it over $\mathbb{F}_p$. We prove it as a consequence of a more general statement.

Lemma 3. Let $\mathbb{U}$ be the finite field with p^n elements, and $\mathbb{U}'/\mathbb{U}$ an extension field with $[\mathbb{U}' : \mathbb{U}] = p^i$. Let $\alpha \in \mathbb{U}'$ be such that

$$\mathrm{Tr}_{\mathbb{U}'/\mathbb{U}}(\alpha) = \beta \neq 0, \tag{2}$$

then $\mathbb{F}_p[\beta] \subset \mathbb{F}_p[\alpha]$ and p^i divides $[\mathbb{F}_p[\alpha] : \mathbb{F}_p[\beta]]$.

Proof. Equation (2) can be written as $\beta = \sum_j \alpha^{p^{jn}}$, thus $\mathbb{F}_p[\beta] \subset \mathbb{F}_p[\alpha]$. The rest of the proof follows by induction on i. If $[\mathbb{U}' : \mathbb{U}] = 1$, then $\alpha = \beta$ and there is nothing to prove. If $i \geq 1$, let $\mathbb{U}''$ be the intermediate extension such that $[\mathbb{U}' : \mathbb{U}''] = p$ and let $\alpha' = \mathrm{Tr}_{\mathbb{U}'/\mathbb{U}''}(\alpha)$, then, by $\mathbf{P}_2$, $\mathrm{Tr}_{\mathbb{U}''/\mathbb{U}}(\alpha') = \beta$ and by induction hypothesis p^{i-1} divides $[\mathbb{F}_p[\alpha'] : \mathbb{F}_p[\beta]]$.

Now, suppose that p does not divide $[\mathbb{F}_p[\alpha] : \mathbb{F}_p[\alpha']]$. Since $\mathbb{F}_p[\alpha'] \subset \mathbb{U}''$, this implies that p does not divide $[\mathbb{U}''[\alpha] : \mathbb{U}'']$; but $\alpha \in \mathbb{U}'$ and $[\mathbb{U}' : \mathbb{U}''] = p$ by construction, so necessarily $[\mathbb{U}''[\alpha] : \mathbb{U}''] = 1$ and $\alpha \in \mathbb{U}''$. This implies $\mathrm{Tr}_{\mathbb{U}'/\mathbb{U}''}(\alpha) = p\alpha = 0$ and, by $\mathbf{P}_2$, $\beta = 0$. Thus, we have a contradiction and p must divide $[\mathbb{F}_p[\alpha] : \mathbb{F}_p[\alpha']]$. The claim follows. $\square$

Corollary 4. With the same notation as above, if $\mathrm{Tr}_{\mathbb{U}'/\mathbb{U}}(\alpha)$ generates $\mathbb{U}$ over $\mathbb{F}_p$, then $\mathbb{F}_p[\alpha] = \mathbb{U}'$.

Hereafter, recall that we write $\gamma_i = G_i \bmod K_i$. We prove that the γ_i's meet the conditions of the corollary.

Lemma 5. If $p \neq 2$, for $i \geq 0$, $\mathbb{U}_i$ is a field and, for $i \geq 1$, $\mathrm{Tr}_{\mathbb{U}_i/\mathbb{U}_{i-1}}(\gamma_i) = -\gamma_{i-1}$.

Proof. Induction on i: for $i = 0$, this is true by hypothesis. For $i \geq 1$, by induction hypothesis $\mathbb{U}_0, \ldots, \mathbb{U}_{i-1}$ are fields; we then set $i' = i - 1$ and prove by nested induction that $\mathrm{Tr}_{\mathbb{U}_{i'}/\mathbb{F}_p}(\gamma_{i'}) \neq 0$ under the hypothesis that $\mathbb{U}_0, \ldots, \mathbb{U}_{i'}$ are fields. This, by (Lidl and Niederreiter, 1996, Th. 2.25), implies that $X_i^p - X_i - \gamma_{i-1}$ is irreducible in $\mathbb{U}_{i-1}[X_{i+1}]$ and $\mathbb{U}_i$ is a field.

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 and we are done. For $i' \geq 1$, 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 the equality $\mathrm{Tr}_{\mathbb{U}_{i'}/\mathbb{F}_p}(\gamma_{i'}) = -\mathrm{Tr}_{\mathbb{U}_{i'-1}/\mathbb{F}_p}(\gamma_{i'-1})$. The induction assumption implies that this is non-zero, and the claim follows. $\square$

Lemma 6. If $p = 2$, for $i \geq 0$, $\mathbb{U}_i$ is a field. For $i \geq 2$, $\mathrm{Tr}_{\mathbb{U}_i/\mathbb{U}_{i-1}}(\gamma_i) = 1 + \gamma_{i-1}$ and

$$\mathrm{Tr}_{\mathbb{U}_1/\mathbb{U}_0}(\gamma_1) = \begin{cases} 1 + \gamma_0 & \text{if } d \text{ even,} \\ 1 & \text{if } d \text{ odd.} \end{cases}$$

Proof. The proof closely follows the previous one. 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 $i' = 1$ and d odd, $\mathrm{Tr}_{\mathbb{U}_1/\mathbb{U}_0}(\gamma_1) = \mathrm{Tr}_{\mathbb{U}_1/\mathbb{U}_0}(x_1) = 1$ by $\mathbf{P}_3$, and $\mathrm{Tr}_{\mathbb{U}_0/\mathbb{F}_p}(1) = d \bmod 2 \neq 0$. For all the other cases

$$\gamma_{i'} = x_{i'}^2 x_{i'} = \gamma_{i'-1} + (1 + \gamma_{i'-1})x_{i'},$$

thus $\mathrm{Tr}_{\mathbb{U}_{i'}/\mathbb{U}_{i'-1}}(\gamma_{i'}) = 1 + \gamma_{i'-1}$ by $\mathbf{P}_3$ and $\mathrm{Tr}_{\mathbb{U}_{i'-1}/\mathbb{F}_p}(1) = 0$. In any case, using the induction hypothesis and $\mathbf{P}_2$, we conclude $\mathrm{Tr}_{\mathbb{U}_{i'}/\mathbb{F}_p}(\gamma_{i'}) = 1$ and this concludes the proof. $\square$

Proof of Theorem 2. If $p \neq 2$, by Lemma 5 and $\mathbf{P}_2$, $\mathrm{Tr}_{\mathbb{U}_i/\mathbb{U}_0}(\gamma_i) = (-1)^i\gamma_0$, thus $\mathbb{U}_i = \mathbb{F}_p[\gamma_i]$ by Corollary 4 and the fact that $\gamma_0 = x_0$ generates $\mathbb{U}_0$ over $\mathbb{F}_p$.

If $p = 2$, we first prove that $\mathbb{U}_1 = \mathbb{F}_p[\gamma_1]$. If d is odd, $\gamma_1^p + \gamma_1 = x_0$ implies $\mathbb{U}_0 \subset \mathbb{F}_p[\gamma_1]$, but $\gamma_1 \notin \mathbb{U}_0$, thus necessarily $\mathbb{U}_1 = \mathbb{F}_p[\gamma_1]$. If d is even, $\mathrm{Tr}_{\mathbb{U}_1/\mathbb{U}_0}(\gamma_1) = 1 + \gamma_0$ clearly generates $\mathbb{U}_0$ over $\mathbb{F}_p$, thus $\mathbb{U}_1 = \mathbb{F}_p[\gamma_1]$ by Corollary 4. Now we proceed like in the $p \neq 2$ case by observing that $\mathrm{Tr}_{\mathbb{U}_i/\mathbb{U}_1}(\gamma_i) = 1 + \gamma_1$ generates $\mathbb{U}_1$ over $\mathbb{F}_p$.

Now, for any p, the theorem follows since clearly $\mathbb{F}_p[\gamma_i] \subset \mathbb{F}_p[x_i]$. $\square$

Remark that the choice of the tower of Theorem 2 is in some sense *optimal* between the choices given by Corollary 4. In fact, each of the G_i's is the "simplest" polynomial in $\mathbb{F}_p[X_i]$ such that $\mathrm{Tr}_{\mathbb{U}_i/\mathbb{F}_p}(\gamma_i) \neq 0$, in terms of lowest degree and least number of monomials.

We furthermore remark that the construction we made in this section gives us a family of normal elements for free. In fact, recall the following proposition from (Hachenberger, 1997, Section 5).

Proposition 7. Let $\mathbb{U}'/\mathbb{U}$ be an extension of finite fields with $[\mathbb{U}' : \mathbb{U}] = kp^i$ where k is prime to p and let $\mathbb{U}''$ be the intermediate field of degree k over $\mathbb{U}$. Then $x \in \mathbb{U}'$ is normal over $\mathbb{U}$ if and only if $\mathrm{Tr}_{\mathbb{U}'/\mathbb{U}''}(x)$ is normal over $\mathbb{U}$. In particular, if $[\mathbb{U}' : \mathbb{U}] = p^i$, then $x \in \mathbb{U}'$ is normal over $\mathbb{U}$ if and only if $\mathrm{Tr}_{\mathbb{U}'/\mathbb{U}}(x) \neq 0$.

Then we easily deduce the following corollary.

Corollary 8. Let $(\mathbb{U}_0, \ldots, \mathbb{U}_k)$ be an Artin-Schreier tower defined by some $(G_i)_{0 \leq i < k}$. Then, every γ_i is normal over $\mathbb{U}_0$; furthermore γ_i is normal over $\mathbb{F}_p$ if and only if $\mathrm{Tr}_{\mathbb{U}_i/\mathbb{U}_0}(\gamma_i)$ is normal over $\mathbb{F}_p$.

In the construction of Theorem 2, if we furthermore suppose that γ_0 is normal over $\mathbb{F}_p$, using Lemma 5 we easily see that the conditions of the corollary are met for $p \neq 2$. For $p = 2$, this is the case only if $[\mathbb{U}_0 : \mathbb{F}_p]$ is even (we omit the proofs that if γ_0 is normal then so are $-\gamma_0$ and $1 + \gamma_0$).

Remark. Observe however that this does not imply the normality of the x_i's. In fact, they can *never* be normal because $\mathrm{Tr}_{\mathbb{U}_i/\mathbb{U}_{i-1}}(x_i) = 0$ by $\mathbf{P}_3$. Granted that γ_0 is normal over $\mathbb{F}_p$, it would be interesting to have an efficient algorithm to switch representations from the univariate $\mathbb{F}_p$-basis in x_i to the $\mathbb{F}_p$-normal basis generated by γ_i.

3.2. Building the tower

This subsection introduces the basic algorithms required to build the tower, that is, compute the required minimal polynomials Q_i.

Composition. We give first 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 the algorithm NaiveCompose.

Lemma 9. NaiveCompose has cost $O(\deg(P)^2 k\ell)$.

Proof. At step i, ρ and S have degree at most ik. Computing the sum $S + p_i\rho$ takes $O(ik)$ operations and computing the product ρR takes $O(ik\ell)$ operations, since R has ℓ terms. The total cost of step i is thus $O(ik\ell)$, whence a total cost of $O(\deg(P)^2 k\ell)$. $\square$

NaiveCompose

```
Input $P, R \in \mathbb{F}_p[X]$.
Output $P(R)$.
  (1) write $P = \sum_{i=0}^{\deg(P)} p_i X^i$, with $p_i \in \mathbb{F}_p$;
  (2) let $S = 0$, $\rho = 1$;
  (3) for $i \in [0, \dots, \deg(P)]$, let $S = S + p_i\rho$ and $\rho = \rho R$;
  (4) return $S$.
```

Compose

```
Input $P, R \in \mathbb{F}_p[X]$.
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 $i \in [0, \dots, c]$, let $Q_i =$ Compose$(P_i, R)$;
  (5) let $Q = 0$;
  (6) for $i \in [c, \dots, 0]$, let $Q = QR(X^{p^n}) + Q_i$;
  (7) return $Q$.
```

Theorem 10. If R has degree k and ℓ non-zero coefficients and if $\deg(P) = s$, then $\mathsf{Compose}(P, R)$ outputs $P(R)$ in $O(ps\log_p(s)k\ell)$ operations.

Proof. To analyze the cost, we let $\mathsf{K}(c, n)$ be the cost of Compose when $\deg(P) \le (c+1)p^n$, with $c < p$. Then $\mathsf{K}(c, 0) \in O(c^2k\ell)$. For $n > 0$, at each pass in the loop at step 6, $\deg(Q) < cp^nk$, so that the multiplication (using the naive algorithm) and addition take $O(cp^nk\ell)$ operations. Thus the cost of the loop is $O(c^2p^nk\ell)$, and the total cost satisfies

$$\mathsf{K}(c, n) \le (c + 1)\mathsf{K}(p - 1, n - 1) + O(c^2p^nk\ell).$$

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

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

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

A binary divide-and-conquer algorithm (von zur Gathen and Gerhard, 1999, 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 2 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 a construction by Cantor (1989), 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 2. 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 10 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 10. The same arguments hold for $i = 2$ when $p = 2$ and d is odd.

To deal with other indices 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 11. 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 $F_1, \dots, F_e$ 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 $j \le e$, $Q_j^\star = Q^\star \bmod F_j$ is in $\mathbb{F}_p[Y]$ and independent from j; the claim follows by Chinese Remaindering.

For $j \le e$, 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 $j, j' \le e$, $a_j = a_{j'}^k$ for some k coprime to $2p-1$, so that $Q_j^\star = Q_{j'}^\star$, as needed.

To conclude, note that for $j \le e$, $Q_j^\star(a_j Y) = Q_j^\star(Y)$, 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 (Cantor, 1989): 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 12. Given Q_i, one can compute Q_{i+1} in $O(p^{i+2} d \log_p(p^i d) + \mathsf{M}(p^{i+2}d)\log(p))$ operations.

Proof. Let $A = \mathbb{F}_p[X]/\Phi$. The algorithm by Brent (1993) computes Φ in $O(p^2)$ operations; then, polynomial multiplications in degree s in $A[Y]$ can be done in $O(\mathsf{M}(sp))$ operations by Kronecker substitution. The overall cost of computing $Q_i^\star$ is $O(\mathsf{M}(p^{i+2}d)\log p)$ using (von zur Gathen and Gerhard, 1999, Algo. 10.3). To get Q_{i+1} we use algorithm Compose with $R = Y^p - Y$, which costs $O(p^{i+2} d \log_p(p^i d))$. □

The former cost is linear in $p^{i+2}d$, up to logarithmic factors, for an input of size $p^i d$ 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 2, $\mathbb{U}_i$ 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 $X_{i-1} > X_i$:

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

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

$$\begin{aligned} \mathbf{D}_i &= \left(x_{i-1}^a x_i^b \mid 0 \le a < p^{i-1}d,\ 0 \le b < p\right), \\ \mathbf{C}_i &= \left(x_i^a \mid 0 \le a < p^i d\right). \end{aligned} \tag{3}$$

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

$$v = v_0 + v_1 x_i + \cdots + v_{p-1} x_i^{p-1}, \quad \text{with all } v_j \vdash \mathbb{U}_{i-1} \tag{4}$$

and $\mathsf{Lift\text{-}up}$ does the opposite.

Hereafter, we let $\mathsf{L} : \mathbb{N} - \{0\} \to \mathbb{N}$ be such that both $\mathsf{Push\text{-}down}$ and $\mathsf{Lift\text{-}up}$ can be performed in $\mathsf{L}(i)$ operations; to simplify some expressions appearing later on, we add the mild constraints that $p\,\mathsf{L}(i) \le \mathsf{L}(i+1)$ and $p\,\mathsf{M}(p^i d) \in O(\mathsf{L}(i))$. 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 13. One can take $\mathsf{L}(i)$ in $O(p^{i+1} d \log_p^2(p^i d) \ + \ p\,\mathsf{M}(p^i d))$.

Remark that the input and output have size $p^i d$; 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. $\mathsf{Push\text{-}down}$ is a divide-and-conquer process, adapted to the shape of our tower; $\mathsf{Lift\text{-}up}$ uses classical ideas of trace computations (as in the algorithm $\mathsf{FindParameterization}$ of Section 2.3); the values we need will be obtained using the transposed version of $\mathsf{Push\text{-}down}$.

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

$$R_n = \textstyle\sum_{j=0}^{n-1} Y^{p^j}. \tag{5}$$

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 $\ell \le p^{n-1}$, 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 $(X_i^j, \dots, Y^{k-1}X_i^j)_{j<p}$ and rows by $(X_i^j, \dots, Y^{k+p^{n-1}-1}X_i^j)_{j<p}$. 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 $O(pnk)$ operations.

4.2. Push-down

The input of $\mathsf{Push\text{-}down}$ is $v \vdash \mathbb{U}_i$, that is, given in the basis $\mathbf{C}_i$; we see it as a polynomial $V \in \mathbb{F}_p[X_i]$ of degree less than $p^i d$. 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 14. We have $\deg(W, Y) \le \deg(V)/p$.

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 $X_i^{pk}, \dots, X_i^{pk+p-1}$ modulo $X_i^p - X_i - Y$ have degree at most k in Y. $\square$

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

$$V = V_0 + V_1 X_i^{p^n} + \cdots + V_c X_i^{cp^n},$$

with all V_j in $\mathbb{F}_p[X_i]$ of degree less than p^n. First, we recursively reduce $V_0, \dots, V_c$ modulo $X_i^p - X_i - Y$, to obtain bivariate polynomials $W_0, \dots, W_c$. Let R_n be the polynomial defined in Equation (5). 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 $\mathsf{Compose}$. 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 $v \vdash \mathbb{U}_i$.
Output $v$ written as $v_0 + \cdots + v_{p-1}x_i^{p-1}$ with $v_j \vdash \mathbb{U}_{i-1}$.
  (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 15. Algorithm Push-down is correct and takes $O(p^{i+1} d \log_p^2(p^i d) + p\,\mathsf{M}(p^i d))$ operations.

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 cost of that loop is $O(nc^2p^n)$. As in Theorem 10, we deduce that the cost of Push-down-rec is $O(n^2c^2p^n)$.

In Push-down, we have $cp^n < p^i d$ and $n < \log_p(p^i d)$, so the previous cost is seen to be $O(p^{i+1} d \log_p^2(p^i d))$. Reducing one coefficient of Z modulo Q_{i-1} takes $O(\mathsf{M}(p^i d))$ operations, 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

Before giving the details for Lift-up, 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 (Bostan, Lecerf, and Schost, 2003), 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 (Bostan, Lecerf, and Schost, 2003, Sect. 5.2): the transposed Euclidean division amounts to compute the values of a sequence linearly generated by the polynomial Q_{i-1} from its first $p^{i-1}d$ values.

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 = $ 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$^*(M, c, n)$.
```

4.4. *Lift-up*

Let v be given in 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 the values of the trace $\mathrm{Tr}_{\mathbb{U}_i/\mathbb{F}_p}$ on the basis $\mathbf{D}_i$ are known. The latter will be given under the form of the (finite) generating series

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

see the discussion below.

Then, as in Subsection 2.3, we use trace formulas to write v as a polynomial in x_i: we see $\mathbb{U}_i$ as a separable extension over $\mathbb{F}_p$ and we look for a parameterization $v = A(x_i)$. To do this, we compute the values of $L = v \cdot \mathrm{Tr}_{\mathbb{U}_i/\mathbb{F}_p}$ on the basis $\mathbf{D}_i$ via transposed multiplication (see Subsection 2.3) and rewrite equations (1) as

$$M = \sum_{j<p^i d} L(x_i^j) X_i^j, \quad N = M\,\mathrm{rev}_{p^i d}(Q_i) \bmod X_i^{p^i d}. \tag{6}$$

To compute the values $L(x_i^j)$ for $0 \le j < p^i d$ we could use a naive algorithm, as in step 2 of FindParameterization, or use (Shoup, 1994, Th. 4); it is however more efficient to use Push-down* as it was shown in the previous subsection. The rest of the computation goes as in steps 3 and 4 of FindParametrization.

Proposition 16. Algorithm Lift-up is correct and takes $O(p^{i+1} d \log_p^2(p^i d) + p\,\mathsf{M}(p^i d))$ operations.

Lift-up

```
Input  $v$ written as $v_0 + \cdots + v_{p-1}x_i^{p-1}$ with $v_j \vdash \mathbb{U}_{i-1}$.
Output $v \vdash \mathbb{U}_i$.
  (1) let $W$ be the canonical preimage of $v$ in $\mathbb{F}_p[X_{i-1}, X_i]$;
  (2) let $L = \mathsf{TransposedMul}(W, S_i)$;
  (3) let $M = \mathsf{Push\text{-}down}^*(L)$;
  (4) let $N = M\,\mathrm{rev}_{p^i d}(Q_i) \bmod X_i^{p^i d}$;
  (5) let $V = \mathrm{rev}_{p^i d-1}(N){Q'_i}^{-1} \bmod Q_i$;
  (6) return the residue class of $V$ modulo $Q_i$.
```

Proof. Correctness is clear by the discussion above. TransposedMul implements the transposed multiplication; Pascal and Schost (2006, Coro. 2) give an algorithm of cost $O(\mathsf{M}(p^i d))$ for this. The last subsection showed that step 3 has the same cost as Push-down. Then, the costs of steps 4 and 5 are $O(\mathsf{M}(p^i d))$ and step 6 is free since V is reduced. $\square$

Propositions 15 and 16 prove Theorem 13. 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 $O(\mathsf{M}(p^i d))$ operations by (Pascal and Schost, 2006, Prop. 8). Then, we need ${Q'_i}^{-1} \bmod Q_i$; this takes $O(\mathsf{M}(p^i d)\log(p^i d))$ operations 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 (2000). 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 $\mathrm{T}_{p^j d} : x \mapsto \sum_{\ell=0}^{p^j d-1} x^{p^\ell}$. First, however, we describe how to compute values of the iterated Frobenius operator $x \mapsto x^{p^n}$ by a recursive descent in the tower.

We focus on computing the iterated Frobenius for $n < d$ or $n = p^j d$. In both cases, similarly to (5), we have:

$$x_i^{p^n} = x_i + \beta_{i-1,n}, \quad \text{with} \quad \beta_{i-1,n} = \mathrm{T}_n(\gamma_{i-1}). \tag{7}$$

Assuming $\beta_{i-1,n}$ is known, the recursive step of the Frobenius algorithm follows: starting from $v \vdash \mathbb{U}_i$, we first write $v = v_0 + \cdots + v_{p-1}x_i^{p-1}$, with $v_h \vdash \mathbb{U}_{i-1}$; by (7) and the linearity of the Frobenius, we deduce that

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

Then, we compute all $v_h^{p^n}$ recursively; the final sum is computed using Horner's scheme. Remark that this variant is not limited to the case where $n < d$ or of the form $p^j d$: an arbitrary n would do as well. However, we impose this limitation since these are the only values we need to compute $\mathrm{T}_{p^j d}$.

In the case $n = p^j d$, any $v \in \mathbb{U}_j$ is left invariant by this Frobenius map, thus we stop the recursion when $i = j$, as there is nothing left to do. In the case $n < d$, we stop the recursion when $i = 0$ and apply (von zur Gathen and Shoup, 1992, Algorithm 5.2). We summarize the two variants in one unique algorithm IterFrobenius.

IterFrobenius

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

As mentioned above, the algorithm requires the values $\beta_{i',n}$ 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.

Theorem 17. On input $v \vdash \mathbb{U}_i$ and $n = p^j d$, algorithm IterFrobenius correctly computes v^{p^n} and takes $O((i-j)\mathsf{L}(i))$ operations.

Proof. Correctness is clear. We write $\mathsf{F}(i,j)$ for the complexity on inputs as in the statement; then $\mathsf{F}(0,j) = \cdots = \mathsf{F}(j,j) = 0$ because step 1 comes at no cost. For $i > j$, each pass through step 6 involves a multiplication by $x_i + \beta_{i-1,n}$, of cost of $O(p\mathsf{M}(p^{i-1}d))$, assuming $\beta_{i-1,n} \vdash \mathbb{U}_{i-1}$ is known. Altogether, we deduce the recurrence relation

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

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

Theorem 18. On input $v \vdash \mathbb{U}_i$ and $n < d$, algorithm IterFrobenius correctly computes v^{p^n} and takes $O(p^i\mathsf{C}(d)\log(n) + i\mathsf{L}(i))$ operations.

Proof. The analysis is identical to the previous one, except that step 2 is now executed instead of step 1 and this costs $O(\mathsf{C}(d)\log(n))$ by (von zur Gathen and Shoup, 1992, Lemma 5.3). The conclusion follows by observing that step 2 is repeated p^i times. □

Next, we compute pseudotraces. We use the following relations, whose verification is straightforward:

$$\mathrm{T}_{n+m}(v) = \mathrm{T}_n(v) + \mathrm{T}_m(v)^{p^n}, \qquad \mathrm{T}_{nm}(v) = \sum_{h=0}^{m-1} \mathrm{T}_n(v)^{p^{hn}}.$$

We give two *divide-and-conquer* algorithms that do a slightly different *divide* step; each of them is based on one of the previous formulas. The first one, LittlePseudotrace, is meant to compute T_d. It follows a binary divide-and-conquer scheme similar to (von zur Gathen and Shoup, 1992, Algorithm 5.2). The second one, Pseudotrace, computes $\mathrm{T}_{p^j d}$ for $j > 0$. It uses the previous formula with $n = p^{j-1}d$ and $m = p$, computing Frobenius-es for such n; when $j = 0$, it invokes the first algorithm.

LittlePseudotrace

```
Input $v$, $i$, $n$ with $v \vdash \mathbb{U}_i$ and $0 < n \leq d$.
Output $T_n(v) \vdash \mathbb{U}_i$.
  (1) if $n = 1$ return $v$;
  (2) let $m = \lfloor n/2 \rfloor$;
  (3) let $t =$ LittlePseudotrace($v$, $i$, $m$);
  (4) let $t = t+$ IterFrobenius($t$, $i$, $m$);
  (5) if $n$ is odd, let $t = t+$ IterFrobenius($v$, $i$, $n$);
  (6) return $t$
```

Pseudotrace

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

Theorem 19. Algorithm LittlePseudotrace is correct and takes

$$O(p^i \mathsf{C}(d) \log^2(n) + i\mathsf{L}(i) \log(n))$$

operations.

Proof. Correctness is clear. For the cost analysis, we write $\mathsf{PT}(i, n)$ for the cost on input i and n, so $\mathsf{PT}(i, 1) = O(1)$. For $n > 1$, step 3 costs $\mathsf{PT}(i, \lfloor n/2 \rfloor)$, steps 4 and 5 cost both $O(p^i \mathsf{C}(d) \log^2(n) + i\mathsf{L}(i))$ by Theorem 18. This gives $\mathsf{PT}(i, n) = \mathsf{PT}(i, \lfloor n/2 \rfloor) + O(p^i \mathsf{C}(d) \log^2(n) + i\mathsf{L}(i))$, and thus $\mathsf{PT}(i, n) \in O(p^i \mathsf{C}(d) \log^2(n) + i\mathsf{L}(i) \log n)$. □

Theorem 20. Algorithm Pseudotrace is correct and takes

$$\mathsf{PT}(i) = O((pi + \log(d))i\mathsf{L}(i) + p^i \mathsf{C}(d) \log^2(d)) \tag{8}$$

operations for $j \leq i$.

Proof. Correctness is clear. For the cost analysis, we write $\mathsf{PT}(i, j)$ for the cost on input i and j, so theorem 19 gives $\mathsf{PT}(i, 0) = O(p^i \mathsf{C}(d) \log^2(d) + i\mathsf{L}(i) \log(d))$. For $j > 0$, step 2 costs $\mathsf{PT}(i, j-1)$, step 3 costs $O(pi\mathsf{L}(i))$ by Theorem 17 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 $O(p^{i+2}d + p^i \mathsf{C}(d))$, up to logarithmic factors, for an input and output size of $p^i d$: this time, due to modular compositions in $\mathbb{U}_0$, the cost is not linear in d.

Finally, let us discuss precomputations. On input v, i, d, the algorithm $\mathsf{LittlePseudotrace}$ makes less than $2\log d$ calls to $\mathsf{IterFrobenius}(x,i,n)$ for some value $x \in \mathbb{U}_i$ and for $n \in N$ where the set N only depends on d. When we construct $\mathbb{U}_{i+1}$, we compute (only) all $\beta_{i,n} = \mathrm{T}_n(\gamma_i) \vdash \mathbb{U}_i$, for increasing $n \in N$, using the $\mathsf{LittlePseudotrace}$ algorithm. The inner calls to $\mathsf{IterFrobenius}$ only use pseudotraces that are already known. Besides, a single call to $\mathsf{LittlePseudotrace}(\gamma_i, i, d)$ actually computes *all* $\mathrm{T}_n(\gamma_i)$ in $O(p^i\mathsf{C}(d)\log^2 d + i\mathsf{L}(i)\log d)$ operations. Same goes for the precomputation of all $\beta_{i,p^jd} = \mathrm{T}_{p^jd}(\gamma_i) \vdash \mathbb{U}_i$, for $j \le i$, using the $\mathsf{Pseudotrace}$ algorithm: this costs $\mathsf{PT}(i)$. Observe that in total we only store $O(k^2 + k\log d)$ elements of the tower, thus the space requirements are quasi-linear.

Remark. A dynamic programming version of $\mathsf{LittlePseudotrace}$ as in (von zur Gathen and Shoup, 1992, Algorithm 5.2) would only precompute $\beta_{i,2^e}$ for $2^e < d$, thus reducing the storage from $2\log d$ to $\lfloor \log d \rfloor$ elements. This would also allow to compute T_n for any $n < d$ without needing any further precomputation. Using this algorithm and a decomposition of $n > d$ as $n = r + \sum_j c_j p^j d$ with $r < d$ and $c_j < p$, one could also compute T_n and x^{p^n} at essentially the same cost. We omit these improvements since they are not essential to the next Section.

6. Arbitrary towers

Finally, we bring our previous algorithms to an arbitrary tower, using the isomorphism algorithm by Couveignes (2000). 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 $\mathbb{U}_0 = \mathbb{F}_p[X_0]/Q_0$; as already remarked in Section 3.2, if p does not divide d we can easily lift the restriction on the trace of Q_0 by composing Q_0 with $X_0 - 1$.[1] We let $(G_i)_{0\le i<k}$ and $(\mathbb{U}_0, \ldots, \mathbb{U}_k)$ be as in Section 3.

We also consider another sequence $(G'_i)_{0\le i<k}$, that defines another tower $(\mathbb{U}'_0, \ldots, \mathbb{U}'_k)$ with $\mathbb{U}_0 = \mathbb{U}'_0 = \mathbb{F}_p[x_0]$. 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 $\mathbb{U}'_i$ in the basis

$$\mathbf{B}'_i = \left({x_0}^{e_0} {x'_1}^{e_1} \cdots {x'_i}^{e_i} \;\middle|\; \text{for } 0 \le e_0 < d \text{ and } 0 \le e_j < p \text{ for } 1 \le j \le i \right).$$

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 $(x_0, x'_1 \ldots, x'_i)$, with $s_i \vdash \mathbb{U}_i$ (obviously, we take $s_0 = x_0$). This isomorphism, denoted by $\sigma_{\mathbf{s}_i}$, takes as input v written in the basis $\mathbf{B}'_i$ and outputs $\sigma_{\mathbf{s}_i}(v) \vdash \mathbb{U}_i$.

To analyze costs, we use the functions L and PT introduced, respectively, in Theorem 13 and Equation 8. We also let $2 \le \omega \le 3$ be a feasible exponent for linear algebra over $\mathbb{F}_p$ (von zur Gathen and Gerhard, 1999, Ch. 12).

Theorem 21. Given Q_0 and $(G'_i)_{0\le i<k}$, one can find $\mathbf{s}_k = (s_0, \ldots, s_k)$ in $O(d^\omega k + \mathsf{PT}(k) + \mathsf{M}(p^{k+1}d)\log(p))$ operations. Once they are known, one can apply $\sigma_{\mathbf{s}_k}$ and $\sigma_{\mathbf{s}_k}^{-1}$ using $O(k\,\mathsf{L}(k))$ operations.

[1] When $p|d$, this restriction is more serious: we can lift it by taking a finite field isomorphism, but this comes at the cost of having to compute the isomorphism, and eventually loosing the benefit of an optimized arithmetic for $\mathbb{U}_0$ (e.g., when Q_0 is sparse). Hence we prefer keeping the restriction and leaving to the reader the task of adapting our construction to the more general setting.

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 $\alpha \vdash \mathbb{U}_i$, 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. Couveignes (2000) 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

$$X^{p^{p^{i-1}d}} - X = \eta, \quad \text{with} \quad \eta = \mathrm{T}_{p^{i-1}d}(\alpha). \tag{9}$$

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

$$X^p - X - \alpha + \mu^p - \mu. \tag{10}$$

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 (9).

For this purpose, observe that the left hand side of (9) is $\mathbb{U}_{i-1}$-linear and its matrix in the basis $(1, \dots, x_i^{p-1})$ is

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

Then, algorithm ApproximateAS finds the required solution.

ApproximateAS

```
Input $\eta \vdash \mathbb{U}_i$ such that (9) has a solution.
Output $\mu \vdash \mathbb{U}_i$ solution of (9).
(1) let $\eta_0 + \eta_1 x_i + \cdots + \eta_{p-2}x_i^{p-2} = $ Push-down($\eta$);
(2) for $j \in [p-1, \dots, 1]$,
        let $\mu_j = \frac{1}{jT}\left(\eta_{j-1} - \sum_{h=j+1}^{p-1}\binom{h}{j-1}\beta_{i-1,p^{i-1}d}^{h-j+1}\mu_h\right)$;
(3) return Lift-up($\mu_1 x_i + \ldots + \mu_{p-1}x_i^{p-1}$).
```

Theorem 22. Algorithm ApproximateAS is correct and takes $O(\mathsf{L}(i))$ operations.

Proof. Correctness is clear from Gaussian elimination. For the cost analysis, remark that $\beta_{i-1,p^{i-1}d}$ has already been precomputed as a prerequisite for the iterated Frobenius and pseudotrace algorithms. 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. $\square$

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

Theorem 23. Algorithm Artin-Schreier is correct and takes $O(d^\omega + \mathsf{PT}(i))$ operations, where $\mathsf{PT}(i) = O((pi + \log(d))i\mathsf{L}(i) + p^i\mathsf{C}(d)\log^2(d))$ as in Theorem 20.

Proof. Correctness follows from the previous discussion. For the complexity, let $\mathsf{AS}(i)$ be the cost for $\alpha \vdash \mathbb{U}_i$. 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 $i \geq 1$, 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 $O(\mathsf{PT}(i))$. $\square$

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 in the multivariate basis $\mathbf{B}'_i$ of $\mathbb{U}'_i$; the output is $t = \sigma_{\mathbf{s}_i}(v) \vdash \mathbb{U}_i$. As before, the algorithm is recursive: we write $v = \Sigma_{j<p} v_j(x_0, \ldots, x'_{i-1}){x'_i}^j$, whence

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

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 $t \vdash \mathbb{U}_i$, 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 $t_j \vdash \mathbb{U}_{i-1}$. Next, we rewrite this as $t = t'_0 + \cdots + t'_{p-1}s_i^{p-1}$, with all $t'_j \vdash \mathbb{U}_{i-1}$, 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 FindParameterization mentioned in Subsection 2.3, in the extension $\mathbb{U}_i = \mathbb{U}_{i-1}[X_i]/P_i$.

Proposition 24. Algorithms ApplyIsomorphism and ApplyInverse are correct and both take $O(i\mathsf{L}(i))$ operations.

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 FindParameterization of Subsection 2.3. By using Kronecker substitution to

ApplyIsomorphism

```
Input $v, i$ with $v \in \mathbb{U}'_i$ written in the basis $\mathbf{B}'_i$.
Output $\sigma_{\mathbf{s}_i}(v) \vdash \mathbb{U}_i$.
  (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 $s_{i,0} + \cdots + s_{i,p-1} x_i^{p-1} =$ Push-down($s_i$);
  (4) for $j \in [0, \ldots, p-1]$ let $t_j =$ 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 Lift-up($t$).
```

ApplyInverse

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

multiply elements in the basis $\mathbf{D}_i$, the cost of both is $O(p\mathsf{M}(p^i d))$, which is in $O(\mathsf{L}(i))$ by assumption on L. We conclude as in Theorem 17. □

6.3. Proof of Theorem 21

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 24, we can compute $\alpha = \sigma_{\mathbf{s}_{i-1}}(\gamma'_{i-1}) \vdash \mathbb{U}_{i-1}$ in $O((i-1)\mathsf{L}(i-1)) \subset O(i\mathsf{L}(i))$ operations. Applying a lift-up to α, we are then in the conditions of Theorem 23, 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 24.

7. Experimental results

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

Implementation. We packaged the algorithms of this paper in a `C++` library called `FAAST` and made it available under the terms of the `GNU GPL` software license from `http://www.lix.polytechnique.fr/Labo/Luca.De-Feo/FAAST/`.

`FAAST` is implemented on top of the `NTL` library by Shoup (2003) which provides the basic univariate polynomial arithmetic needed here. Our library handles three NTL classes of finite fields: `GF2` for $p = 2$, `zz_p` for word-size p and `ZZ_p` for arbitrary p; this choice is made by the user at compile-time through the use of `C++` templates and the

$$\begin{tikzcd}
 & v \vdash \mathbb{U}_k \arrow[dr, dashed, tail, bend left] & \\
\mathbb{U}'_k \arrow[r, dash, "\sigma'"] & \mathbb{U}_k \arrow[r, dash, "\sigma''"] \arrow[d, dash] & \mathbb{U}''_k \arrow[dll, dashed, dash] \\
\mathbb{U}'_{k-1} \arrow[r, dash, "\sigma'"] \arrow[drr, dashed, dash] & \mathbb{U}_{k-1} \arrow[r, dash, "\sigma''"] \arrow[d, dotted, dash] & \mathbb{U}''_{k-1} \\
\mathbb{U}'_1 \arrow[r, dash, "\sigma'"] & \mathbb{U}_1 \arrow[r, dash, "\sigma''"] \arrow[d, dash] & \mathbb{U}''_1 \arrow[d, dashed] \\
 & \mathbb{U}_0 & v \vdash \mathbb{U}_0[x''_1, \ldots, x'_{k-1}, x''_k]
\end{tikzcd}$$

Fig. 1. An example of conversion from the univariate basis to the multivariate basis of the tower $(\mathbb{U}_0, \mathbb{U}''_1, \ldots, \mathbb{U}'_{k-1}, \mathbb{U}''_k)$.

resulting code is thus quite efficient. Optionally, `NTL` can be combined with the `gf2x` package by Brent, Gaudry, Thomé, and Zimmermann (2008) for better performance in the $p = 2$ case, as we did in our experiments.

All the algorithms of Sections 3–5 are faithfully implemented in `FAAST`. The algorithms ApplyIsomorphism and ApplyInverse have slightly different implementations that allow more flexibility. Instead of being recursive algorithms doing the change to and from the multivariate basis $\mathbf{B}'_i = ({x_0}^{e_0} {x'_1}^{e_1} \cdots {x'_i}^{e_i})$, they only implement the change to and from the bivariate basis

$$\mathbf{D}'_i = \left({x_{i-1}}^{e_{i-1}} {x'_i}^{e_i} \;\middle|\; \text{for } 0 \le e_{i-1} < p^{i-1}d \text{ and } 0 \le e_i < p \right).$$

Equivalently, this amounts to switch between the representations

$$\vdash \mathbb{U}_i \quad \text{and} \quad \vdash \mathbb{U}_{i-1}[X'_i]/(X_i'^p - X'_i - \gamma'_{i-1}).$$

The same result as one call to ApplyIsomorphism or ApplyInverse can be obtained by i calls to the routines `toUnivariate()` and `toBivariate()` respectively. However, in the case where several generic Artin-Schreier towers, say $(\mathbb{U}'_0, \ldots, \mathbb{U}'_k)$ and $(\mathbb{U}''_0, \ldots, \mathbb{U}''_k)$, are built using the algorithms of Section 6, this allows to *mix* the representations by letting the user chose to switch to any of the bases $(x_0^{e_0} y_1^{e_1} \cdots y_i^{e_i})$ where y_i is either x'_i or x''_i. In other words this allows the user to *zig-zag* in the lattice of finite fields as in Figure 1.

Besides the algorithms presented in this paper, `FAAST` also implements some algorithms described in (De Feo, 2011) for minimal polynomials, evaluation and interpolation, as they are required for the isogeny computation algorithm.

Experimental results. We compare our timings with those obtained in Magma 2.16 (Bosma, Cannon, and Playoust, 1997) for similar questions. All results are obtained on an Intel Xeon E5520 (2.26GHz). Our experiments revealed a regression in the performances of Magma 2.16, concerning one algorithm. When such difference is noticeable, we also plot the timings obtained with Magma 2.11 on an equivalent machine (Intel Xeon E5430).

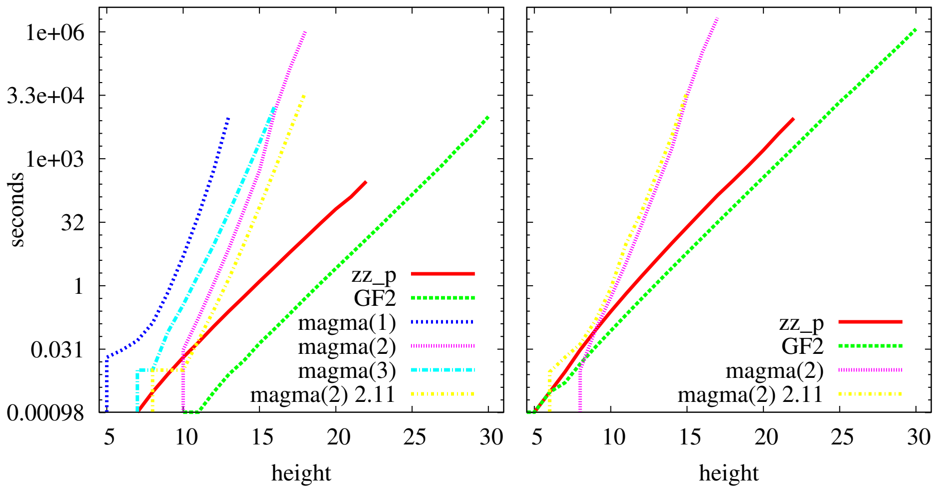

Fig. 2. Build time (left) and isomorphism time (right) with respect to tower height. Plot is in logarithmic scale.

The experiments for the `FAAST` library were only made for the classes `GF2` and `zz_p`. The class `ZZ_p` was left out because all the primes that can be reasonably handled by our library fit in one machine-word. 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 $P \in k[X]$ (written magma(2));
- `ext<k|p>` builds an extension of degree p of k (written magma(3)).

We made experiments for each of these choices where this makes sense.

The parameters to our algorithms are (p, d, k). Thus, our experiments describe the following situations:

- *Increasing the height k.* Here we take $p = 2$ and $d = 1$ (that is, $\mathbb{U}_0 = \mathbb{F}_2$); the x-coordinate gives the number of levels we construct and the y-coordinate gives timings in seconds, in *logarithmic* scale.

  This is done in Figure 2. We let the height of the tower increase and we give timings for (1) building the tower of Section 3 and (2) computing an isomorphism with a random arbitrary tower as in Section 6. In the latter experiment, only the magma(2) approach was meaningful for Magma.
- *Increasing the degree d of $\mathbb{U}_0$.* Here we take $p = 5$ and we construct 2 levels; the x-coordinate gives the degree $d = [\mathbb{U}_0 : \mathbb{F}_p]$ and the y-coordinate gives timings in seconds. This is done in Figure 3 (left).
- *Increasing p.* Here we take $d = 1$ (thus $\mathbb{U}_0 = \mathbb{F}_p$) and we construct 2 levels; the x-coordinate gives the characteristic p and the y-coordinate gives timings in seconds. This is done in Figure 3 (right).

The timings of our code are significantly better for increasing height or increasing d. Not surprisingly, for increasing p, the magma(1) approach performs better than any other:

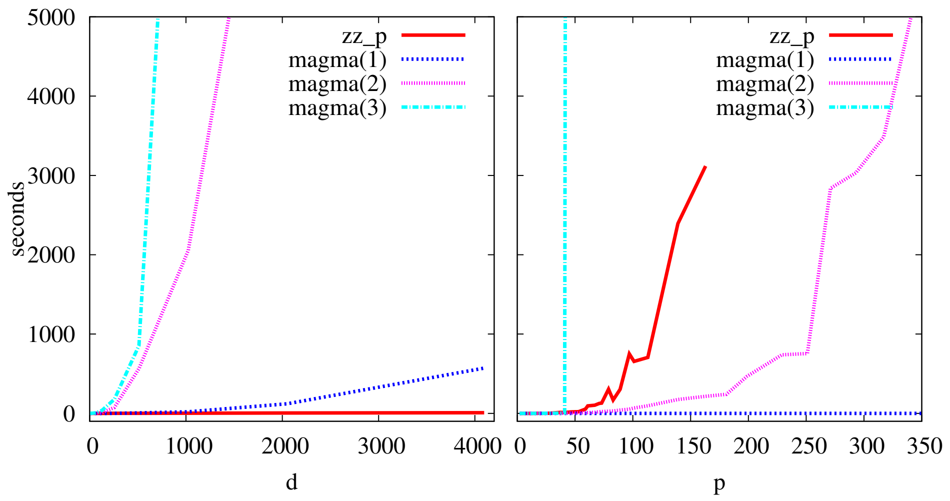

Fig. 3. Build times with respect to d (left) and p (right).

level	Primitive	Push-d.	Lift-up	Product	Reciprocal	apply σ^{-1}	apply σ
19	1.143	0.304	1.265	0.039	0.649	0.652	1.290
20	2.566	0.609	2.796	0.081	1.544	1.314	2.602
21	5.686	1.225	6.147	0.187	3.598	2.409	2.668
22	12.660	2.515	13.746	0.463	8.355	5.565	11.179
23	28.511	5.295	31.200	1.046	19.522	12.323	24.740

Table 1. Some timings in seconds for arithmetics in a generic tower built over $\mathbb{F}_2$ using `GF2`.

the `quo` operation simply creates a residue class ring, regardless of the (ir)reducibility of the modulus, so the timing for building two levels barely depend on p. The most adapted approach for this situation presumably is magma(2); yet we notice that `FAAST` has reasonable performances for characteristics up to about $p = 50$.

In Tables 1 and 2 we provide some comparative timings for the different arithmetic operations provided by `FAAST`. The column "Primitive" gives the time taken to build one level of the primitive tower (this includes the precomputation of the data as described in Subsection 4.4); the other entries are self-explanatory. Product and inversion are just wrappers around `NTL` routines: in these operations we did not observe any overhead compared to the native `NTL` code.

Finally, we mention the cost of precomputation. The precomputation of the images of σ as explained in Section 6 is quite expensive; most of it is spent computing pseudotraces. Indeed it took one week to precompute the data in Figure 2 (right), while all the other data can be computed in a few hours. There is still space for some minor improvement in `FAAST`, mainly tweaking recursion thresholds and implementing better algorithms for small and moderate input sizes. Still, we think that only a major algorithmic improvement could consistently speed up this phase.

level	Primitive	Push-d.	Lift-up	Product	Reciprocal	apply σ^{-1}	apply σ
18	13.618	0.884	13.712	0.476	10.753	1.337	3.578
19	30.288	1.814	30.432	1.001	23.046	2.850	7.798
20	65.632	3.953	66.889	2.106	51.544	6.564	18.141
21	128.190	8.347	131.271	4.791	121.349	14.396	39.296
22	296.671	11.396	298.541	6.413	249.520	28.851	86.628

Table 2. Some timings in seconds for arithmetics in a generic tower built over $\mathbb{F}_2$ using `zz_p`.

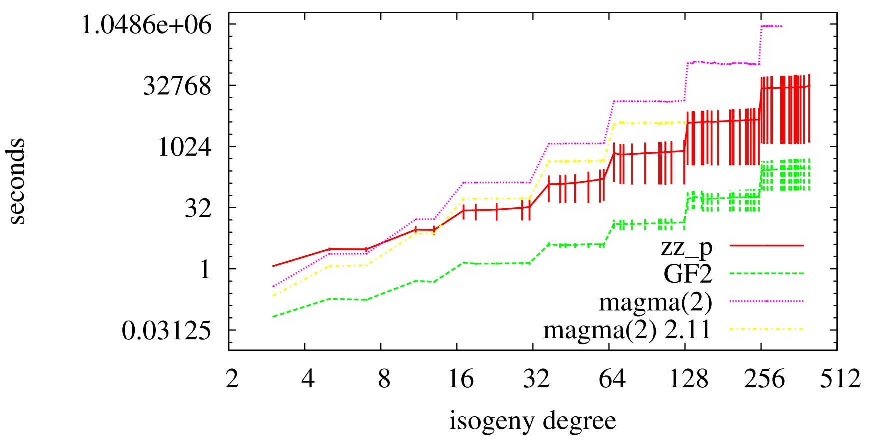

Fig. 4. Timings for the isogeny algorithm. Isogenies of degree increasing degree are computed between curves defined over $\mathbb{F}_{2^{101}}$.

Isogeny algorithm. An isogeny is a regular map between two elliptic curves $\mathscr{E}$ and $\mathscr{E}'$ that is also a group morphism. In cryptology, isogenies are used in the Schoof-Elkies-Atkin point-counting algorithm (see Blake et al., 1999), but also in more recent constructions (e.g. Rostovtsev and Stolbunov, 2006; Teske, 2006), and the fast computation of isogenies remains a difficult challenge.

Our interest here is the isogeny algorithm by Couveignes (1996), which computes isogenies of degree $\sim p^k$; the algorithm relies on the interpolation of a rational function at special points in an Artin-Schreier tower. The original algorithm by Couveignes (1996) was first implemented by Lercier (1997); later, Couveignes (2000) described improvements to speed up the computation, but as we already mentioned, a key component, fast arithmetic in Artin-Schreier towers, was still missing. Recently De Feo (2011) has combined this paper's algorithms and other improvements to achieve a completely explicit version of (Couveignes, 2000).

The algorithm is composed of 5 phases:

(1) Depending on the degree ℓ of the isogeny to be computed, a parameter k is chosen such that $p^{k-1}(p-1) > 4\ell - 2$;
(2) a primitive tower of height $\sim k$ is computed (the precise height depends on $\mathscr{E}$ and $\mathscr{E}'$, in the example of figure 4 it is always equal to $k-2$);

degree	step 2	step 3	step 5	step 6		
				preconditioning	avg # iterations	iteration
3	0.004	0.013	0.036	0.001	4	0.0000
5	0.002	0.041	0.092	0.004	8	0.0002
11	0.004	0.122	0.229	0.019	16	0.0003
17	0.006	0.354	0.554	0.094	32	0.0010
37	0.014	0.985	1.320	0.227	64	0.0039
67	0.038	2.662	3.100	1.133	128	0.0131
131	0.076	7.006	7.219	6.117	256	0.0477
257	0.185	17.931	16.606	34.524	512	0.1731

Table 3. Comparative timings for each phase of the isogeny algorithm using `GF2`.

(3) an Artin-Schreier tower in which the p^k-torsion points of $\mathscr{E}$ are defined is computed and an isomorphism is constructed to the primitive tower;
(4) an Artin-Schreier tower in which the p^k-torsion points of $\mathscr{E}'$ are defined is computed and an isomorphism is constructed to the primitive tower;
(5) a mapping from $\mathscr{E}[p^k]$ to $\mathscr{E}'[p^k]$ is computed through interpolation;
(6) all the possible mappings from $\mathscr{E}[p^k]$ to $\mathscr{E}'[p^k]$ are computed through modular composition until one is found that yields an isogeny.

We ran experiments for curves defined over the base field $\mathbb{F}_{2^{101}}$ for increasing isogeny degree. Figure 4 shows the timings for two implementations of (De Feo, 2011) based on `FAAST` and one implementation of the same algorithm based on the magma(2) approach; remark that the time scale is logarithmic. The running time is probabilistic because step 6 stops as soon as it has found an isogeny; we plot the average running times with bars around them for minimum/maximum times; the distribution is uniform. Note that the plot in the original ISSAC '09 version of this paper shows timings that are one order of magnitude worse. This was due to a bug that has later been fixed.

Table 3 shows comparative timings for each phase of the algorithm. The reason why we left step 4 out of the table is that it is essentially the same as step 3 and timings are nearly identical. Step 6 is asymptotically the most expensive one; it uses some preconditioning to speed up each iteration of the loop. From the point of view of this paper, the most interesting steps are 2-5 since they are the only ones that make use of the library `FAAST`.

For $p = 2$, it should be noted that the isogeny algorithm by Lercier (1996) has better performance; for generic, small, p we mention as well a new algorithm by Lercier and Sirvent (2008). See (De Feo, 2010, 2011) for further discussions on isogeny computation.

Acknowledgements

We would like to thank J.-M. Couveignes and F. Morain for useful discussions.

References

Blake, I. F., Seroussi, G., Smart, N. P., 1999. Elliptic curves in cryptography. Cambridge University Press, New York, NY, USA.

Bosma, W., Cannon, J., Playoust, C., 1997. The MAGMA algebra system I: the user language. Journal of Symbolic Computation 24 (3-4), 235–265.

Bostan, A., Lecerf, G., Schost, E., 2003. Tellegen's principle into practice. In: ISSAC '03: Proceedings of the 2003 international symposium on Symbolic and algebraic computation. ACM, New York, NY, USA, pp. 37–44.

Brent, R. P., 1993. On computing factors of cyclotomic polynomials. Mathematics of Computation 61 (203), 131–149.

Brent, R. P., Gaudry, P., Thomé, E., Zimmermann, P., 2008. Faster multiplication in GF(2)[x]. In: ANTS-VIII'08: Proceedings of the 8th international conference on Algorithmic number theory. Springer-Verlag, Berlin, Heidelberg, pp. 153–166.

Bürgisser, P., Clausen, M., Shokrollahi, M. A., Feb. 1997. Algebraic Complexity Theory. Vol. 315 of A Series of Comprehensive Studies in Mathematics. Springer.

Cantor, D. G., 1989. On arithmetical algorithms over finite fields. Journal of Combinatorial Theory, Series A 50 (2), 285–300.

Couveignes, J.-M., 1996. Computing l-Isogenies using the p-Torsion. In: ANTS-II: Proceedings of the Second International Symposium on Algorithmic Number Theory. Springer-Verlag, London, UK, pp. 59–65.

Couveignes, J.-M., 2000. Isomorphisms between Artin-Schreier towers. Mathematics of Computation 69 (232), 1625–1631.

De Feo, L., Dec. 2010. Algorithmes Rapides pour les Tours de Corps Finis et les Isogénies. Ph.D. thesis, Ecole Polytechnique X.

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

De Feo, L., Schost, E., 2009. Fast arithmetics in Artin-Schreier towers over finite fields. In: ISSAC '09: Proceedings of the 2009 international symposium on Symbolic and algebraic computation. ACM, New York, NY, USA, pp. 127–134.

Hachenberger, D., 1997. Finite fields: normal bases and completely free elements. Kluwer Academic Pub.

Kaltofen, E., Jun. 2000. Challenges of symbolic computation: My favorite open problems. Journal of Symbolic Computation 29 (6), 891–919.

Kedlaya, K. S., Umans, C., 2008. Fast modular composition in any characteristic. In: FOCS '08: Proceedings of the 2008 49th Annual IEEE Symposium on Foundations of Computer Science. IEEE Computer Society, Washington, DC, USA, pp. 146–155.

Lercier, R., 1996. Computing isogenies in GF(2,n). In: ANTS-II: Proceedings of the Second International Symposium on Algorithmic Number Theory. Springer-Verlag, London, UK, pp. 197–212.

Lercier, R., Jun. 1997. Algorithmique des courbes elliptiques dans les corps finis. Ph.D. thesis, LIX – CNRS.

Lercier, R., Sirvent, T., 2008. On Elkies subgroups of ℓ-torsion points in elliptic curves defined over a finite field. Journal de théorie des nombres de Bordeaux 20 (3), 783–797.

Li, X., Moreno Maza, M., Schost, E., 2007. Fast arithmetic for triangular sets: from theory to practice. In: ISSAC '07: Proceedings of the 2007 international symposium on Symbolic and algebraic computation. ACM, New York, NY, USA, pp. 269–276.

Lidl, R., Niederreiter, H., Oct. 1996. Finite Fields (Encyclopedia of Mathematics and its Applications). Cambridge University Press.

Mateer, T., 2008. Fast Fourier transform algorithms with applications. Ph.D. thesis, Clemson University, Clemson, SC, USA.

Pascal, C., Schost, E., 2006. Change of order for bivariate triangular sets. In: ISSAC '06: Proceedings of the 2006 international symposium on Symbolic and algebraic computation. ACM, New York, NY, USA, pp. 277–284.

Rostovtsev, A., Stolbunov, A., Apr. 2006. Public-key cryptosystem based on isogenies. Cryptology ePrint Archive, Report 2006/145.

Rouillier, F., May 1999. Solving Zero-Dimensional systems through the rational univariate representation. Applicable Algebra in Engineering, Communication and Computing 9 (5), 433–461.

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

Shoup, V., 1999. Efficient computation of minimal polynomials in algebraic extensions of finite fields. In: ISSAC '99: Proceedings of the 1999 international symposium on Symbolic and algebraic computation. ACM, New York, NY, USA, pp. 53–58.

Shoup, V., 2003. NTL: A library for doing number theory. `http://www.shoup.net/ntl`.

Teske, E., Jan. 2006. An elliptic curve trapdoor system. Journal of Cryptology 19 (1), 115–133.

von zur Gathen, J., Gerhard, J., 1999. Modern computer algebra. Cambridge University Press, New York, NY, USA.

von zur Gathen, J., Gerhard, J., 2002. Polynomial factorization over GF(2). Mathematics of Computation 71 (240), 1677–1698.

von zur Gathen, J., Shoup, V., 1992. Computing Frobenius maps and factoring polynomials. Computational Complexity 2, 187–224.

Wang, Y., Zhu, X., Apr. 1988. A fast algorithm for the Fourier transform over finite fields and its VLSI implementation. IEEE Journal on Selected Areas in Communications 6 (3), 572–577.