Fast arithmetics in Artin-Schreier towers over finite fields

L. De Feo\textsuperscript{1} and É. Schost\textsuperscript{2}

\textsuperscript{1}École Polytechnique and INRIA, France
\textsuperscript{2}ORCCA and CSD, The University of Western Ontario, London, ON

July 31, 2009
ISSAC, Seoul, Korea
From crypto to computer algebra

\[ \mathbb{U}_k \rightarrow - - E[p^k] \]

\[ p \]

\[ \mathbb{U}_{k-1} \rightarrow - - E[p^{k-1}] \]

\[ p \]

\[ \mathbb{U}_2 \rightarrow - - E[p^2] \]

\[ p \]

\[ \mathbb{U}_1 \rightarrow - - E[p] \]

\[ \mathbb{F}_q \]

\[ p^k \]-torsion points of elliptic curves

\[ E : y^2 = x^3 + ax + b \quad a, b \in \mathbb{F}_q \]

\[ p^k \]-torsion points are not necessarily defined in the base field. We want to:

- compute primitive \( p^k \)-torsion points,
- apply Galois actions on them,
- evaluate maps between elliptic curves,
- ...

Applications

- Isogeny computation [Couveignes '96].
- \( p \)-torsion points of generic abelian varieties;
**Definition (Artin-Schreier polynomial)**

For a field $K$ of characteristic $p$, $\alpha \in K$,

$$X^p - X - \alpha$$

is an Artin-Schreier polynomial.

**Theorem**

If $K$ is finite, $X^p - X - \alpha$ irreducible $\iff$ $\text{Tr}_{K/F_p}(\alpha) \neq 0$.

If $\eta \in K$ is a root, then $\eta + 1, \ldots, \eta + (p - 1)$ are roots.

**Definition (Artin-Schreier extension)**

If $\mathcal{P}$ is an irreducible Artin-Schreier polynomial, $\mathbb{L} = \mathbb{K}[X]/\mathcal{P}(X)$.

$L/K$ is called an Artin-Schreier extension.
Our context

\[ U_k = \frac{U_{k-1}[X_k]}{P_{k-1}(X_k)} \]

\[ U_1 = \frac{U_0[X_1]}{P_0(X_1)} \]

\[ U_0 = \mathbb{F}_{p^d} = \frac{\mathbb{F}_p[X_0]}{Q(X_0)} \]

Towers over finite fields

\[ P_i = X^p - X - \alpha_i \]

We say that \((U_0, \ldots, U_k)\) is defined by \((\alpha_0, \ldots, \alpha_{k-1})\) over \(U_0\).

ANY separable extension of degree \(p\) can be expressed this way.
Size, complexities

\[ \#U_i = p^{p^i} \]

**Optimal representation**

All common representations achieve it: \( O(p^i d) \)

**Complexities**

<table>
<thead>
<tr>
<th>Type</th>
<th>Complexity</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>optimal</td>
<td>( O(p^i d) )</td>
<td>addition</td>
</tr>
<tr>
<td>quasi-optimal</td>
<td>( \tilde{O}(i^a p^i d) )</td>
<td>FFT multiplication</td>
</tr>
<tr>
<td>almost-optimal</td>
<td>( \tilde{O}(i^a p^{i+b} d) )</td>
<td></td>
</tr>
<tr>
<td>suboptimal</td>
<td>( \tilde{O}(i^a p^{i+b} d^c) )</td>
<td>naive multiplication</td>
</tr>
<tr>
<td>too bad</td>
<td>( \tilde{O}(i^a (p^{i+b})^{e} d^c) )</td>
<td></td>
</tr>
</tbody>
</table>

**Multiplication function \( M(n) \)**

FFT: \( M(n) = O(n \log n \log \log n) \),  
Naive: \( M(n) = O(n^2) \).
Outline

1. Representation
2. Arithmetics
3. Implementation and benchmarks
Representation matters!

Multivariate representation of \( v \in \mathbb{U}_i \)

\[
v = X_0^{d-1}X_1^{p-1} \cdots X_i^{p-1} + 2X_0^{d-1}X_1^{p-1} \cdots X_i^{p-2} + \cdots
\]

Univariate representation of \( v \in \mathbb{U}_i \)

- \( \mathbb{U}_i = \mathbb{F}_p[x_i] \),
- \( v = c_0 + c_1x_i + c_2x_i^2 + \cdots + c_{p^id-1}x_i^{p^id-1} \) with \( c_i \in \mathbb{F}_p \).

How much does it cost to...

- Multiply?
- Express the embedding \( \mathbb{U}_{i-1} \subset \mathbb{U}_i \)?
- Express the vector space isomorphism \( \mathbb{U}_i = \mathbb{U}_i^{p-1} \)?
- Switch between the representations?
A primitive tower

Definition (Primitive tower)

A tower is primitive if \( U_i = \mathbb{F}_p[X_i] \).

In general this is not the case. Think of \( P_0 = X^p - X - 1 \).

Theorem (extends a result in [Cantor ’89])

Let \( x_0 = X_0 \) such that \( \text{Tr}_{U_0/\mathbb{F}_p}(x_0) \neq 0 \), let

\[
\begin{align*}
P_0 &= X^p - X - x_0 \\
P_i &= X^p - X - x_i^{2^{p-1}}
\end{align*}
\]

with \( x_{i+1} \) a root of \( P_i \) in \( U_{i+1} \).

Then, the tower defined by \((P_0, \ldots, P_{k-1})\) is primitive.

Some tricks to play when \( p = 2 \).
Computing the minimal polynomials

We look for $Q_i$, the minimal polynomial of $x_i$ over $\mathbb{F}_p$

Algorithm [Cantor ’89]

- $Q_0 = Q$  
- $Q_1 = Q_0(X^p - X)$

Let $\omega$ be a $2p - 1$-th root of unity,

- $q_{i+1}(X^{2p-1}) = \prod_{j=0}^{2p-2} Q_i(\omega^j X)$  
- $Q_{i+1} = q_{i+1}(X^p - X)$

Complexity

$O(M(p^{i+2}d) \log p)$
Outline

1 Representation

2 Arithmetics

3 Implementation and benchmarks
Level embedding

**Push-down**

**Input** \( v \in \mathbb{U}_i \),

**Output** \( v_0, \ldots, v_{p-1} \in \mathbb{U}_{i-1} \) such that \( v = v_0 + \cdots + v_{p-1}x_i^{p-1} \).

**Lift-up**

**Input** \( v_0, \ldots, v_{p-1} \in \mathbb{U}_{i-1} \),

**Output** \( v \in \mathbb{U}_i \) such that \( v = v_0 + \cdots + v_{p-1}x_i^{p-1} \).

**Complexity function \( L(i) \)**

It turns out that the two operations lie in the same complexity class, we note  \( L(i) \) for it:

\[
L(i) = O (pM(p^i d) + p^{i+1} d \log_p (p^i d)^2)
\]
Level embedding

Change of order

\[
\begin{align*}
X_i^p - X_i - X_{i-1}^{2p-1} &= 0 \\
Q_{i-1}(X_{i-1}) &= 0
\end{align*}
\quad \leftrightarrow \quad
\begin{align*}
Q_i(X_i) &= 0 \\
X_{i-1} &= R(X_i)/S(X_i)
\end{align*}
\]

Rational Univariate Representation ([Rouillier '99])

- Push-down: left-to-right,
- Lift-up: right-to-left,
- Going right-to-left = looking for RUR,
- Equivalently, changing order from $X_{i-1} > X_i$ to $X_i > X_{i-1}$.
- Many optimisations for our case.
Push-down

Input $v \vdash U_i$,  
Output $v_0, \ldots, v_{p-1} \vdash U_{i-1}$ s.t. $v = v_0 + \cdots + v_{p-1} x_i^{p-1}$.

1. Reduce $v$ modulo $x_i^p - x_i - x_{i-1}^{2p-1}$ by a divide-and-conquer approach,
2. each of the coefficients of $x_i$ has degree in $x_{i-1}$ less than $2 \deg_{x_i}(v)$,
3. reduce each of the coefficients.
Lift-up

**Power projection**

Let $x$ be fixed. An algorithm that takes a linear form $\ell$ as input and outputs

$$\ell(1), \ell(x), \ldots, \ell(x^n)$$

is said to solve *power projection* problem ([Shoup '99]).

**Trace formulas** [Pascal and Schost '06, Rouillier '99]

- Given $v_0, \ldots, v_{p-1} \in U_{i-1}$,
- $v = v_0 + \cdots + v_{p-1} x_i^{p-1}$ can be recovered using suitable trace formulas.
- Solving them is the power projection problem on input $v \cdot \text{Tr} : x \mapsto \text{Tr}(vx)$.

**Transposed algorithms** (see [Bürgisser, Clausen and Shokrollahi '97])

- *Linear algorithms* can be *transposed* much like linear applications;
- Computing $v \cdot \text{Tr}$ is *transposed multiplication*.
- Computing the power projection for $x_i$ is *transposed push-down*.
Lift-up

\textbf{Input} \quad v_0, \ldots, v_{p-1} \rightarrow U_{i-1}

\textbf{Output} \quad v \rightarrow U_i \quad \text{s.t.} \quad v = v_0 + \cdots + v_{p-1} x_i^{p-1}

1. Compute the linear form \( \text{Tr} \in U_i^{D^*} \),
2. compute \( \ell = (v_0 + \cdots + v_{p-1} x_i^{p-1}) \cdot \text{Tr} \),
3. compute \( P_v = \text{Push-down}^T(\ell) \),
4. compute \( N_v(Z) = P_v(Z) \cdot \text{rev}(Q_i)(Z) \mod Z^{p^i d - 1} \),
5. return \( \text{rev}(N_v)/Q'_i \mod Q_i \).
Speeding up some arithmetics

Divide and conquer

\[ \mathbb{U}_k \]
\[ \mathbb{U}_{k-1} \]
\[ \mathbb{U}_1 \]
\[ \mathbb{U}_0 \]

We improve some operations in \( \mathbb{U}_i \) \( \text{op}(v) \)

Where it works

- traces,
- \( p \)-th roots,
- pseudotrace,
- inversion,
- iterated frobenius,
- ...
Speeding up some arithmetics

Divide and conquer

We improve some operations in $U_i$

- push-down the operands;

$$\text{op}(v) \quad v_0, \ldots, v_{p-1}$$

Where it works

- traces,
- $p$-th roots,
- pseudotraces,
- inversion,
- iterated frobenius,
- ...
Speeding up some arithmetics

Divide and conquer

We improve some operations in $\mathbb{U}_i$

- push-down the operands;
- recursively solve $p$ instances in $\mathbb{U}_{i-1}$;

\[ \text{op}(v) \]
\[ \text{op}(v_0), \ldots, \text{op}(v_{p-1}) \]

Where it works

- traces,
- $p$-th roots,
- pseudotrases,
- inversion,
- iterated frobenius,
- \ldots
Speeding up some arithmetics

Divide and conquer

We improve some operations in $\mathbb{U}_i$

- push-down the operands;
- recursively solve $p$ instances in $\mathbb{U}_{i-1}$;
- combine the results;

$\mathbb{U}_k$ \hspace{2cm} $\mathbb{U}_{k-1}$ \hspace{2cm} $\mathbb{U}_1$ \hspace{2cm} $\mathbb{U}_0$

Where it works

- traces,
- $p$-th roots,
- pseudotraces,
- inversion,
- iterated frobenius,
- ...
Speeding up some arithmetics

**Divide and conquer**

We improve some operations in $\mathbb{U}_i$

- push-down the operands;
- recursively solve $p$ instances in $\mathbb{U}_{i-1}$;
- combine the results;
- lift-up.

$$op(v) \quad op(v_0), \ldots, \quad op(v_{p-1})$$

$$w_0, \ldots, \quad w_{p-1}$$

$$w$$

**Where it works**

- traces,
- $p$-th roots,
- pseudotraces,
- inversion,
- iterated frobenius,
- ...
Important application: Isomorphisms with generic towers

**Generic towers**
- Let \((\alpha_0, \ldots, \alpha_{k-1})\) define a generic tower over \(U_0\).
- if we find an isomorphism we can bring fast arithmetics to it.

**Computing the isomorphism [Couveignes ’00]**

**Goal:** factor \(X^p - X - \alpha_i\) in \(U_{i+1}\).
- Change of variables \(X' = X - \mu\) s.t.
- \(X'^p - X' - \alpha_i\) has a root in \(U_i\).
- Push-down, solve recursively, result is \(\Delta\),
- Lift-up \(\Delta\),
- return \(\Delta + \mu\).
Outline

1 Representation

2 Arithmetics

3 Implementation and benchmarks
Implementation

Implementation in NTL + gf2x

Three types

- GF2: \( p = 2 \), FFT, bit optimisation,
- \( \mathbb{zz}_p \): \( p < 2^{|\text{long}|} \), FFT, no bit-tricks,
- \( \mathbb{ZZ}_p \): generic \( p \), like \( \mathbb{zz}_p \) but slower.

Comparison to Magma

Three ways of handling field extensions

1. \( \text{quo}<U|P> \): quotient of multivariate polynomial ring + Gröbner bases
2. \( \text{ext}<k|P> \): field extension by \( X^p - X - \alpha \), precomputed bases + multivariate
3. \( \text{ext}<k|p> \): field extension of degree \( p \), precomputed bases + multivariate

Benchmarks (on 14 AMD Opteron 2500)

Three modes

- \( p = 2, \ d = 1 \), height varying,
- \( p \) varying, \( d = 1 \), height = 2,
- \( p = 5, \ d \) varying, height = 2.
Construction of the tower + precomputations

![Graph showing construction time vs. height and modulus for different algorithms](image)

- **zz_p**: Green line
- **GF2**: Red line
- **magma(1)**: Blue line
- **magma(2)**: Pink line
- **magma(3)**: Cyan line

The graph illustrates the performance of different algorithms for constructing towers and precomputing operations, with the x-axis representing the height of the tower and the y-axis showing the time in seconds. The modulus p is shown on a separate axis, indicating the scalability of the algorithms with respect to increasing p.
Multiplication

<table>
<thead>
<tr>
<th>height</th>
<th>seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000976562</td>
<td>0.00390625</td>
</tr>
<tr>
<td>0.015625</td>
<td></td>
</tr>
<tr>
<td>0.0625</td>
<td></td>
</tr>
<tr>
<td>0.25</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td></td>
</tr>
<tr>
<td>256</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td></td>
</tr>
<tr>
<td>256</td>
<td></td>
</tr>
<tr>
<td>512</td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td></td>
</tr>
<tr>
<td>2048</td>
<td></td>
</tr>
<tr>
<td>4096</td>
<td></td>
</tr>
</tbody>
</table>

Graphs showing the performance of different implementations over various heights and prime numbers.

L. De Feo and É. Schost ( )
Fast arithmetics in Artin-Schreier towers
ISSAC, July 31, 2009 21 / 24
Isomorphism ([Couveignes ’00] vs Magma)
Benchmarks on isogenies ([Couveignes ’96])

Over $\mathbb{F}_{2^{101}}$, on an Intel Xeon E5430 Quad Core Processor 2.66GHz, 64GB ram

![Graph showing benchmark results for isogeny degree vs. time in seconds.](image-url)
These algorithms are packaged in a library

Download FAAST at
http://www.lix.polytechnique.fr/Labo/Luca.De-Feo/FAAST

We are currently writing an spkg for Sage.