Fast arithmetics in Artin-Schreier towers over finite fields

Luca De Feo\textsuperscript{1}

joint work with É. Schost\textsuperscript{2}

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

October 10, 2009
RAIM, École Normale Supérieure, Lyon
Doing arithmetics in towers of extensions

Standard arithmetics

\[
\begin{align*}
+,-,\times,/ & : \\
(\mathbb{U}_i \times \mathbb{U}_i) & \rightarrow \mathbb{U}_i \\
(u, v) & \mapsto u \text{ op } v
\end{align*}
\]
Doing arithmetics in towers of extensions

Inclusion

\[ \imath : \begin{cases} \mathcal{U}_i & \subset & \mathcal{U}_{i+1} \\ u & \mapsto & \overline{u} \end{cases} \]
Doing arithmetics in towers of extensions

Membership

\[ \ell^{-1} : \left\{ \begin{array}{c}
U_{i+1} \supseteq U_i \\
\ell(v) \mapsto v
\end{array} \right. \]
Doing arithmetics in towers of extensions

Projection

\[
\begin{align*}
\pi &: \left\{ \begin{array}{c}
U_{i+1} \sim U_i^p \simeq U_i[\gamma] \\
v &\mapsto (v_0, \ldots, v_{p-1})
\end{array} \right.
\end{align*}
\]

\[
\pi^{-1} : \left\{ \begin{array}{c}
U_i^p \simeq U_i[\gamma] &\sim \to U_{i+1} \\
(v_0, \ldots, v_{p-1}) &\mapsto \sum_j v_j \gamma^j
\end{array} \right.
\]
Doing arithmetics in towers of extensions

\[ \mathbb{U}_k \quad \mathbb{U}_{k-1} \quad \mathbb{U}_2 \quad \mathbb{U}_1 \quad \mathbb{F}_q \]

Traces

\[ \text{Tr} : \begin{cases} 
\mathbb{U}_{i+1} & \rightarrow & \mathbb{U}_i \\
\nu & \mapsto & \text{Tr}(\nu) 
\end{cases} \]
Doing arithmetics in towers of extensions

\[ \mathbb{U}_k \quad p \quad \mathbb{U}_{k-1} \quad p \quad \mathbb{U}_2 \quad p \quad \mathbb{U}_1 \quad p \quad \mathbb{F}_q \]

Galois action

\[ \varphi : \begin{cases} G \times \mathbb{U}_i & \rightarrow \mathbb{U}_i \\ (\sigma, v) & \mapsto \sigma(v) \end{cases} \]

\[ G := \text{Gal}(\mathbb{U}_{i+1}/\mathbb{U}_i) \cong \mathbb{Z}/p\mathbb{Z} \]
Crypto application: Isogeny computation

\[ \mathbb{U}_{16} \leftarrow - E[2^{18}] \]
\[ \mathbb{U}_{15} \leftarrow - E[2^{17}] \]
\[ \mathbb{U}_{2} \leftarrow - E[16] \]
\[ \mathbb{U}_{1} \leftarrow - E[8] \]
\[ \mathbb{F}_q \leftarrow - E[4] \]

\[ E, E' \text{ elliptic curves with } \#E(F_q) = \#E'(F_q) \]

**Theorem/Algorithm**

Knowing \( E[2^{k+3}] \) and \( E'[2^{k+3}] \)

\[ \Rightarrow \text{all isogenies of degree } < 2^k \]

**Example**

- \( \mathbb{F}_q = \mathbb{F}_{2^{163}} \),
- \( E[4] \subset E(\mathbb{F}_q), \ E[2^{i+2}] \subset E(\mathbb{U}_i) \),
- Isogeny degree \( < 2^{15} \Rightarrow 16 \text{ levels}!! \)
- One element of \( \mathbb{U}_{16} \sim 1.5\text{MB}!! \)
Our context

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

Tower over finite fields

\[ P_i \text{ irreducible polynomial in } U_i[X] \]
Our context

\[ \mathbb{U}_k = \frac{\mathbb{U}_{k-1}[X_k]}{P_{k-1}(X_k)} \]

\[ p \]

\[ \mathbb{U}_{k-1} \]

Tower over finite fields

\[ P_i \text{ irreducible polynomial in } \mathbb{U}_i[X] \]

But this is too hard.
**Artin-Schreier**

**Definition (Artin-Schreier polynomial)**

\( K \) a field of characteristic \( p \), \( \alpha \in K \)

\[ X^p - X - \alpha \]

is an Artin-Schreier polynomial.

**Theorem**

\( K \) 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)**

\( P \) an irreducible Artin-Schreier polynomial.

\[ L = K[X]/P(X). \]

\( L/K \) is called an Artin-Schreier extension.
Our context

\[ \mathbb{U}_k = \frac{\mathbb{U}_{k-1}[X_k]}{P_{k-1}(X_k)} \]

\[ p \]

\[ \mathbb{U}_{k-1} \]

\[ \mathbb{U}_1 = \frac{\mathbb{U}_0[X_1]}{P_0(X_1)} \]

\[ p \]

\[ \mathbb{U}_0 = \mathbb{F}_{p^d} = \frac{\mathbb{F}_p[X_0]}{Q(X_0)} \]

Towers over finite fields

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

We say that \((\mathbb{U}_0, \ldots, \mathbb{U}_k)\) is defined by \((\alpha_0, \ldots, \alpha_{k-1})\) over \(\mathbb{U}_0\).

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

\[ \# \mathbb{U}_i = p^{p^i d} \]

**Optimal representation**

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

**Complexities**

- **optimal:** \( O(p^i d) \) addition
- **quasi-optimal:** \( \tilde{O}(i^a p^i d) \) FFT multiplication
- **almost-optimal:** \( \tilde{O}(i^a p^{i+b} d) \)
- **suboptimal:** \( \tilde{O}(i^a (p^{i+b})^d d^c) \)
- **too bad:** \( \tilde{O}(i^a (p^{i+b})^e d^c) \) naive multiplication

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

- **FFT:** \( M(n) = O(n \log n \log \log n) \),
- **Naive:** \( M(n) = O(n^2) \).
1 Representation

2 More 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** \( v \in \mathbb{U}_i \)

- \( \mathbb{U}_i = \mathbb{F}_p[x_i] \),
- \( v = c_0 + c_1 x_i + c_2 x_i^2 + \cdots + c_{p^i d-1} x_i^{p^i d-1} \) 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 \( \mathbb{U}_i = \mathbb{U}_i^p \)?
- 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

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

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 $p = 2$. 

L. De Feo (École Polytechnique) Fast arithmetics in Artin-Schreier towers RAIM, October 10, 2009 10 / 30
Computing the minimal polynomials

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

Algorithm [Cantor ’89]

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

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

- $q_{i+1}(X^{2p-1}) = \prod_{j=0}^{2p-2} Q_i(\omega^j X)$ not too hard,
- $Q_{i+1} = q_{i+1}(X^p - X)$ easy.

Complexity

$O(M(p^{i+2}d) \log p)$
Yes, we can multiply!

Standard arithmetics

\[
\begin{align*}
&\mathbb{U}_k \\
&\downarrow p \\
&\mathbb{U}_{k-1} \\
&\downarrow p \\
&\mathbb{U}_2 \\
&\downarrow p \\
&\mathbb{U}_1 \\
&\downarrow p \\
&\mathbb{F}_q
\end{align*}
\]

\[\begin{align*}
+,-,\times,\div : \left\{ \begin{array}{c}
\mathbb{U}_i \times \mathbb{U}_i 
\rightarrow \mathbb{U}_i \\
(u,v) \mapsto u \text{ op } v
\end{array} \right. 
\]
Outline

1 Representation

2 More arithmetics

3 Implementation and benchmarks
Level embedding

$$\pi : \begin{cases} U_{i+1} & \sim \rightarrow U_i^p \simeq U_i[\gamma] \\ U_i & \mapsto \rightarrow (v_0, \ldots, v_{p-1}) \end{cases}$$

$$\pi^{-1} : \begin{cases} U_i^p \simeq U_i[\gamma] & \sim \rightarrow U_{i+1} \\ (v_0, \ldots, v_{p-1}) & \mapsto \sum_j v_j \gamma^j$$
**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:

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

Theorem

Up to some simple formulae:

\[
\begin{pmatrix}
\pi^{-1} \\
v
\end{pmatrix}
\sim
\begin{pmatrix}
\pi^T \\
M_v^T \\
\text{Tr}^T
\end{pmatrix}
\]

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

- Tr can be easily computed through the residue formula.
- Linear algorithms can be transposed much like linear applications;
- computing \( v \cdot \text{Tr} := (M_v)(\text{Tr}^T) \) is transposed multiplication.
- Computing \( \pi^T \) is transposed push-down.
**Theorem**

*Up to some simple formulae:*

\[
\begin{pmatrix}
\pi^{-1}
\end{pmatrix}
\begin{pmatrix}
v
\end{pmatrix}
\sim
\begin{pmatrix}
\pi^T
\end{pmatrix}
\begin{pmatrix}
M_v^T
\end{pmatrix}
\begin{pmatrix}
\text{Tr}^T
\end{pmatrix}
\]

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

- Tr can be easily computed through the *residue formula*.
- *Linear algorithms* can be *transposed* much like linear applications;
- computing \( v \cdot \text{Tr} := (M_v)(\text{Tr}^T) \) is *transposed multiplication*.
- Computing \( \pi^T \) is *transposed push-down*. 
Lift-up

Theorem

*Up to some simple formulae:*

\[
\begin{pmatrix}
\pi^{-1}
\end{pmatrix}
\begin{pmatrix} v \end{pmatrix}
\sim
\begin{pmatrix}
\pi^T
\end{pmatrix}
\begin{pmatrix}
M^T_v
\end{pmatrix}
\begin{pmatrix}
\text{Tr}^T
\end{pmatrix}
\]

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

- Tr can be easily computed through the *residue formula*.
- *Linear algorithms* can be *transposed* much like linear applications;
- computing \( v \cdot \text{Tr} := (M_v)(\text{Tr}^T) \) is *transposed multiplication*.
- Computing \( \pi^T \) is *transposed push-down*. 
Lift-up

Theorem

Up to some simple formulae:

\[
\begin{pmatrix}
\pi^{-1}
\end{pmatrix}
\begin{pmatrix} v \\
\end{pmatrix}
\sim
\begin{pmatrix} \pi^T \\
\end{pmatrix}
\begin{pmatrix} M_v^T \\
\end{pmatrix}
\begin{pmatrix} \text{Tr}^T \\
\end{pmatrix}
\]

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

- $\text{Tr}$ can be easily computed through the residue formula.
- Linear algorithms can be transposed much like linear applications;
- computing $v \cdot \text{Tr} := (M_v)(\text{Tr}^T)$ is transposed multiplication.
- Computing $\pi^T$ is transposed push-down.
Lift-up

\begin{itemize}
\item Input $v_0, \ldots, v_{p-1} \vdash \mathbb{U}_{i-1}$
\item Output $v \vdash \mathbb{U}_i$ s.t. $v = v_0 + \cdots + v_{p-1}x_i^{p-1}$
\begin{enumerate}
\item Compute the linear form $\text{Tr} \in \mathbb{U}_i^{D^*}$,
\item compute $\ell = (v_0 + \cdots + v_{p-1}x_i^{p-1}) \cdot \text{Tr}$,
\item compute $P_v = \text{Push-down}^T(\ell)$,
\item compute $N_v(Z) = P_v(Z) \cdot \text{rev}(Q_i)(Z) \mod Z^{p_i d - 1}$,
\item return $\text{rev}(N_v)/Q'_i \mod Q_i$.
\end{enumerate}
\end{itemize}
Speeding up some arithmetics

Galois action

\[ \varphi : \begin{cases} \mathcal{G} \times \mathbb{U}_i & \to \mathbb{U}_i \\ (\sigma, v) & \mapsto \sigma(v) \end{cases} \]

\[ \mathcal{G} := \text{Gal}(\mathbb{U}_{i+1}/\mathbb{U}_i) \simeq \mathbb{Z}/p\mathbb{Z} \]
Speeding up some arithmetics

**Divide and conquer**

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

**Where it works**

- traces,
- $p$-th roots,
- pseudotraces,
- inversion,
- Galois action,
- ...
Speeding up some arithmetics

Divide and conquer

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

- push-down the operands;

$\text{op}(v)$

$\downarrow$

$v_0, \cdots, v_{p-1}$

Where it works

- traces,
- $p$-th roots,
- pseudotraces,
- inversion,
- Galois action,
- \ldots
### Divide and conquer

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

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

\[
\begin{align*}
\mathbb{U}_k & \quad \text{op}(v) \\
\mathbb{U}_{k-1} & \quad \text{op}(v_0), \ldots, \text{op}(v_{p-1}) \\
\mathbb{U}_1 & \\
\mathbb{U}_0 & \\
\end{align*}
\]

### Where it works

- traces,
- $p$-th roots,
- pseudotratces,
- inversion,
- Galois action,
- \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;

\[
\begin{align*}
\mathbb{U}_k & \quad \mathbb{U}_{k-1} \\
\mathbb{U}_1 & \quad \mathbb{U}_0 \\
\text{op}(v) & \quad \text{op}(v_0), \ldots, \text{op}(v_{p-1}) \\
w_0, \ldots, w_{p-1} & \quad w_0, \ldots, w_{p-1}
\end{align*}
\]

**Where it works**

- traces,
- \( p \)-th roots,
- pseudotraces,
- inversion,
- Galois action,
- \( \ldots \)
Speeding up some arithmetics

Divide and conquer

We improve some operations in $U_i$

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

Where it works

- traces,
- $p$-th roots,
- pseudotraces,
- inversion,
- Galois action,
- ...
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. More arithmetics

3. Implementation and benchmarks
Implementation in NTL + gf2x

Three types

- **GF2**: \( p = 2 \), FFT, bit optimisation,
- **zz_p**: \( p < 2^{\text{long}} \), FFT, no bit-tricks,
- **ZZ_p**: generic \( p \), like zz_p but slower.

Comparison to Magma

Three ways of handling field extensions

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

- Time comparison for different heights:
  - `zz_p`: Red line
  - `GF2`: Magenta line
  - `magma(1)`: Blue line
  - `magma(2)`: Light blue line
  - `magma(3)`: Cyan line

- Graphs showing time in seconds for different heights and primes:
  - Vertical axis: Time in seconds
  - Horizontal axis: Height
  - Different graphs for different primes (`p`)

- L. De Feo (École Polytechnique)

[Graphs and data points are shown in the image, illustrating the time taken for various computations.]
Multiplication

0.000976562
0.00390625
0.015625
0.0625
0.25
1
4
16
64
256
10 15 20 25
seconds
height
zz_p
gf2x
magma(1)
magma(2)

0.000976562
0.00390625
0.015625
0.0625
0.25
1
4
16
32 64 128 256 512 1024 2048 4096
seconds
d
zz_p
magma(1)
magma(2)
magma(3)
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

L. De Feo (École Polytechnique)
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.