Fast arithmetics in Artin-Schreier towers over finite fields

Luca De Feo
joint work with É. Schost

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

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

\[ \mathbb{U}_k \]

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

\[ \mathbb{U}_2 \]

\[ \mathbb{U}_1 \]

\[ \mathbb{F}_q \]

\[ +, -, \times, / : \begin{cases} \mathbb{U}_i \times \mathbb{U}_i \to \mathbb{U}_i \\ (u, v) \mapsto u \text{ op } v \end{cases} \]

Standard arithmetics
Doing arithmetics in towers of extensions

\[ U_k \xrightarrow{p} U_{k-1} \]
\[ U_{k-1} \xrightarrow{p} U_{k-2} \]
\[ U_{k-2} \xrightarrow{p} U_{k-3} \]
\[ \cdots \]
\[ F_q \]

Inclusion

\[ \iota : \begin{cases} 
U_i & \subset U_{i+1} \\
\nu & \mapsto \bar{\nu} 
\end{cases} \]
Doing arithmetics in towers of extensions

\[ \mathbb{U}_k \]

\[ p \]

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

\[ \mathbb{U}_2 \]

\[ p \]

\[ \mathbb{U}_1 \]

\[ p \]

\[ \mathbb{F}_q \]

Membership

\[ l^{-1} : \begin{cases} \mathbb{U}_{i+1} \supset \mathbb{U}_i \\ l(v) \mapsto v \end{cases} \]
Doing arithmetics in towers of extensions

Projection

\[ \pi : \begin{cases} 
  \mathbb{U}_{i+1} & \sim \rightarrow \mathbb{U}_i^p \simeq \mathbb{U}_i[\gamma] \\
  v & \mapsto (v_0, \ldots, v_{p-1}) 
\end{cases} \]

\[ \pi^{-1} : \begin{cases} 
  \mathbb{U}_i^p & \simeq \rightarrow \mathbb{U}_i[\gamma] \\
  (v_0, \ldots, v_{p-1}) & \mapsto \sum_j v_j \gamma^j 
\end{cases} \]
Doing arithmetics in towers of extensions

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

Traces

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

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

Galois action

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

\[ G := \text{Gal}(\mathbb{U}_{i+1}/\mathbb{U}_i) \cong \mathbb{Z}/p\mathbb{Z} \]
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 \( \mathbb{U}_{16} \sim 1.5 \text{MB} !! \)

\( E, E' \) elliptic curves
with \( \#E(\mathbb{F}_q) = \#E'(\mathbb{F}_q) \)
Tower over finite fields

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

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

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)**

$\mathbb{K}$ a field of characteristic $p$, $\alpha \in \mathbb{K}$

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

is an Artin-Schreier polynomial.

**Theorem**

$\mathbb{K}$ finite. $X^p - X - \alpha$ irreducible $\iff$ $\text{Tr}_{\mathbb{K}/\mathbb{F}_p}(\alpha) \neq 0$.

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

**Definition (Artin-Schreier extension)**

$\mathcal{P}$ an irreducible Artin-Schreier polynomial.

$$\mathbb{L} = \mathbb{K}[X]/\mathcal{P}(X).$$

$\mathbb{L}/\mathbb{K}$ is called an Artin-Schreier extension.
Our context

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

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

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

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

---

**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: \( O(p^id) \)

**Complexities**

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

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

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

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 \( \mathbb{U}_i = \mathbb{U}_{i-1}^p \)?
- Switch between the representations?
A primitive tower

**Definition (Primitive tower)**

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

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 \( \text{Tr}_{\mathbb{U}_0/\mathbb{F}_p}(x_0) \neq 0 \), let

\[
P_0 = X^p - X - x_0 \\
P_i = X^p - X - x_i^{2^{p-1}}
\]

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

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

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

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

Standard arithmetics

\[ +, -, \times, / : \begin{cases} \mathbb{U}_i \times \mathbb{U}_i & \rightarrow \mathbb{U}_i \\ (u, v) & \mapsto u \text{ op } v \end{cases} \]
Outline

1. Representation

2. More arithmetics

3. Implementation and benchmarks
Level embedding

\[ \pi : \begin{cases} \mathbb{U}_{i+1} \sim \Rightarrow \mathbb{U}_i^p \simeq \mathbb{U}_i[\gamma] \\ \nu \mapsto (\nu_0, \ldots, \nu_{p-1}) \end{cases} \]

\[ \pi^{-1} : \begin{cases} \mathbb{U}_i^p \simeq \mathbb{U}_i[\gamma] \sim \Rightarrow \mathbb{U}_{i+1} \\ (\nu_0, \ldots, \nu_{p-1}) \mapsto \sum_j \nu_j \gamma^j \end{cases} \]
Level embedding

**Push-down**

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

**Lift-up**

**Input** \( v_0, \ldots, v_{p-1} \rightarrow \mathbb{U}_{i-1} \),
**Output** \( v \rightarrow \mathbb{U}_i \) 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:

\[
L(i) = O \left( pM(p^id) + p^{i+1}d \log_p(p^id)^2 \right)
\]
Push-down

---

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

**Output** \( v_0, \ldots, v_{p-1} \uparrow \mathbb{U}_{i-1} \) 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:

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

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

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

- **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{align*}
\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}
\end{align*}

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

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

\textbf{Input} \quad v_0, \ldots, v_{p-1} \rightarrow \mathbb{U}_{i-1} \\
\textbf{Output} \quad v \rightarrow \mathbb{U}_i \quad \text{s.t.} \quad v = v_0 + \cdots + v_{p-1}x_i^{p-1}

1. Compute the linear form \( \text{Tr} \in \mathbb{U}_i^{D^*} \),
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 \( N_v(Z) = P_v(Z) \cdot \text{rev}(Q_i)(Z) \mod Z^{p^id-1} \),
5. return \( \text{rev}(N_v)/Q'_i \mod Q_i \).
Speeding up some arithmetics

\[ \mathbb{U}_k \xrightarrow{p} \mathbb{U}_{k-1} \]

\[ \mathbb{U}_2 \xrightarrow{p} \mathbb{U}_1 \xrightarrow{p} \mathbb{F}_q \]

Galois action

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

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

Divide and conquer

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

Where it works

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

Divide and conquer

We improve some operations in \( U_i \)

- push-down the operands;

\[
\begin{align*}
\text{op}(v) & \quad v_0, \ldots, v_{p-1} \\
\end{align*}
\]

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 \( U_{i-1} \);

\[
\begin{align*}
\text{op}(v) \quad \Downarrow \quad \text{op}(v_0), \ldots, \text{op}(v_{p-1})
\end{align*}
\]

Where it works

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

Divide and conquer

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

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

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;
- recursively solve $p$ instances in $\mathbb{U}_{i-1}$;
- 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 $\mathbb{U}_0$.
- 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 $\mathbb{U}_{i+1}$.
- Change of variables $X' = X - \mu$ s.t.
- $X'^p - X' - \alpha_i$ has a root in $\mathbb{U}_i$.
- Push-down, solve recursively, result is $\Delta$.
- Lift-up $\Delta$.
- Return $\Delta + \mu$. 
Implementation

Implementation in NTL + gf2x

Three types

- GF2: \( p = 2 \), FFT, bit optimisation,
- \( \text{zz}_p \): \( p < 2^{\text{long}} \), FFT, no bit-tricks,
- \( \text{ZZ}_p \): generic \( p \), like \( \text{zz}_p \) 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 + precomputation
Isomorphism ([Couveignes '00] vs Magma)

- $0.000976562$
- $0.03125$
- $1$
- $32$
- $1024$
- $32768$
- $1.04858e+06$
- $5, 10, 15, 20, 25$

seconds vs height

- $zz_p$
- $GF2$
- $magma(2)$

L. De Feo (École Polytechnique)

Fast arithmetics in Artin-Schreier towers

RAIM, October 10, 2009
Benchmarks on isogenies ([Couveignes ’96])

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

![Graph showing the relationship between isogeny degree and execution time for different software tools like magma(2) and $zz^p$.](image)
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.