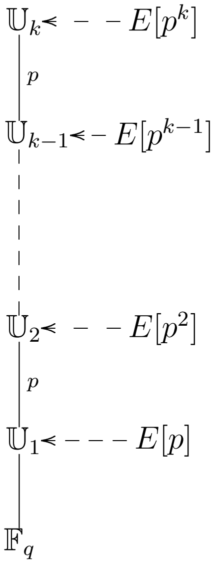
Fast arithmetics in Artin-Schreier towers over finite fields

L. De Feo[1] and É. Schost[2]

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

July 31, 2009
ISSAC, Seoul, Korea

From crypto to computer algebra



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 primtive 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;

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* $\Leftrightarrow \operatorname{Tr}_{\mathbb{K}/\mathbb{F}_p}(\alpha) \neq 0$.
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

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

$$\Big|\, p$$

$$\mathbb{U}_{k-1}$$

$$\vdots$$

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

$$\Big|\, p$$

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

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

$$\#\mathbb{U}_i \; = \; p^{p^i d}$$

$\mathbb{U}_k$

$\mathbb{U}_{k-1}$

$\mathbb{U}_1$

$\mathbb{U}_0$

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:	$\tilde{O}(i^a p^{i+b} d^c)$	
too bad:	$\tilde{O}\left(i^a (p^{i+b})^e d^c\right)$	naive multiplication

Multiplication function $\mathsf{M}(n)$

FFT: $\mathsf{M}(n) = O(n \log n \log\log n)$, Naive: $\mathsf{M}(n) = O(n^2)$.

Outline

Representation matters!

$\mathbb{U}_k$

$\mathbb{U}_{k-1}$

$\mathbb{U}_1$

$\mathbb{U}_0$

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

$\mathbb{U}_k$

$|$

$\mathbb{U}_{k-1}$

$\vdots$

$\mathbb{U}_1$

$|$

$\mathbb{U}_0$

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

$$\begin{aligned} P_0 &= X^p - X - x_0 \\ P_i &= X^p - X - x_i^{2p-1} \end{aligned}$$

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

$\mathbb{U}_k$

$\mathbb{U}_{k-1}$

$\mathbb{U}_1$

$\mathbb{U}_0$

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 ω 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

$$O\left(\mathsf{M}(p^{i+2}d)\log p\right)$$

Outline

Level embedding

$\mathbb{U}_k$

$\mathbb{U}_{k-1}$

$\mathbb{U}_1$

$\mathbb{U}_0$

Push-down

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

Lift-up

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

Complexity function $\mathsf{L}(i)$

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

$$\mathsf{L}(i) = O\left(p\mathsf{M}(p^i d) + p^{i+1} d \log_p (p^i d)^2\right)$$

Level embedding

Change of order

$$\begin{cases} X_i^p - X_i - X_{i-1}^{2p-1} = 0 \\ Q_{i-1}(X_{i-1}) = 0 \end{cases} \quad \leftrightarrow \quad \begin{cases} Q_i(X_i) = 0 \\ X_{i-1} = R(X_i)/S(X_i) \end{cases}$$

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

Push-down

Input $v \dashv \mathbb{U}_i$,
Output $v_0, \ldots, v_{p-1} \dashv \mathbb{U}_{i-1}$ 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

Power projection

Let x be fixed. An algorithm that takes a linear form ℓ 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 $v_0, \ldots, v_{p-1} \dashv \mathbb{U}_{i-1}$,
- $v = v_0 + \cdots + v_{p-1} x_i^{p-1}$ can be recovered using suitable trace formulas.
- Solving them is the power projection problem on input $v \cdot \mathrm{Tr} : x \mapsto \mathrm{Tr}(vx)$.

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

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

Lift-up

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

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

Speeding up some arithmetics

$\mathbb{U}_k$

$\mathbb{U}_{k-1}$

$\mathbb{U}_1$

$\mathbb{U}_0$

Divide and conquer

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

$\mathrm{op}(v)$

Where it works

- traces,
- p-th roots,
- pseudotraces,
- inversion,
- iterated frobenius,
- ...

Speeding up some arithmetics

$\mathbb{U}_k$

$\mathbb{U}_{k-1}$

$\mathbb{U}_1$

$\mathbb{U}_0$

Divide and conquer

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

- push-down the operands;

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

Where it works

- traces,
- p-th roots,
- pseudotraces,
- inversion,
- iterated frobenius,
- ...

Speeding up some arithmetics

$\mathbb{U}_k$ — $\mathbb{U}_{k-1}$ — ⋯ — $\mathbb{U}_1$ — $\mathbb{U}_0$

Divide and conquer

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

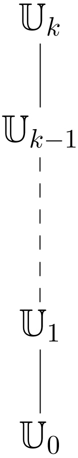
- push-down the operands;
- recursively solve p instances in $\mathbb{U}_{i-1}$;

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

Where it works

- traces,
- p-th roots,
- pseudotraces,
- inversion,
- iterated frobenius,
- ...

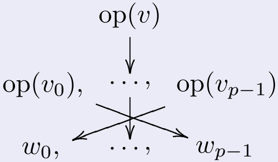
Speeding up some arithmetics



Divide and conquer

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

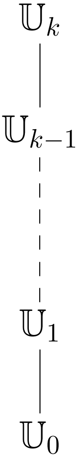
- push-down the operands;
- recursively solve p instances in $\mathbb{U}_{i-1}$;
- combine the results;



Where it works

- traces,
- p-th roots,
- pseudotraces,
- inversion,
- iterated frobenius,
- ...

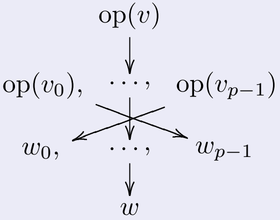
Speeding up some arithmetics



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.



Where it works

- traces,
- p-th roots,
- pseudotraces,
- inversion,
- iterated frobenius,
- ...

Important application : Isomorphisms with generic towers

$\mathbb{U}_k$ — $\mathbb{U}_{k-1}$ — … — $\mathbb{U}_1$ — $\mathbb{U}_0$

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 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 Δ,
- Lift-up Δ,
- return $\Delta + \mu$.

$\mathbb{U}'_k$ — $\mathbb{U}'_{k-1}$ — … — $\mathbb{U}'_1$ — $\mathbb{U}_0$

Outline

Implementation

Implementation in NTL + gf2x

Three types

- `GF2`: $p = 2$, FFT, bit optimisation,
- `zz_p`: $p < 2^{|\texttt{long}|}$, FFT, no bit-tricks,
- `ZZ_p`: generic p, like `zz_p` but slower.

Comparison to Magma

Three ways of handling field extensions

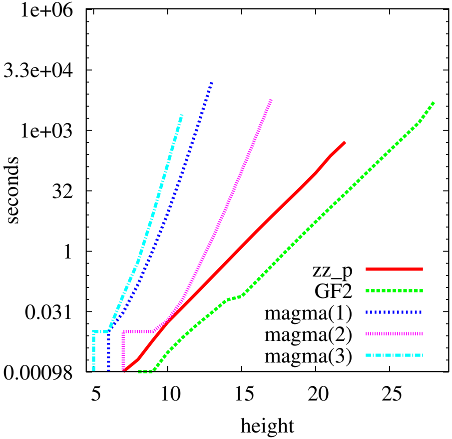
1. `quo<U|P>`: quotient of multivariate polynomial ring + Gröbner bases
2. `ext<k|P>`: field extension by $X^p - X - \alpha$, precomputed bases + multivariate
3. `ext<k|p>`: field extension of degree p, precomputed bases + multivariate

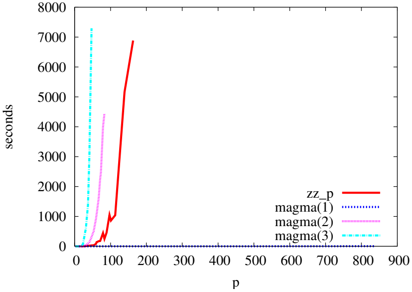
Benchmarks (on 14 AMD Opteron 2500)

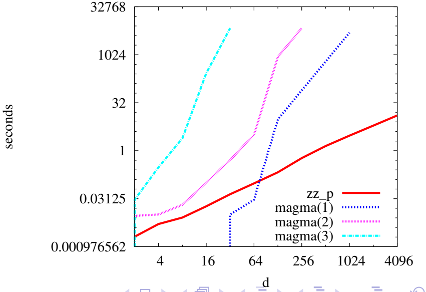
Three modes

- $p = 2$, $d = 1$, height varying,
- p varying, $d = 1$, height $= 2$,
- $p = 5$, d varying, height $= 2$.

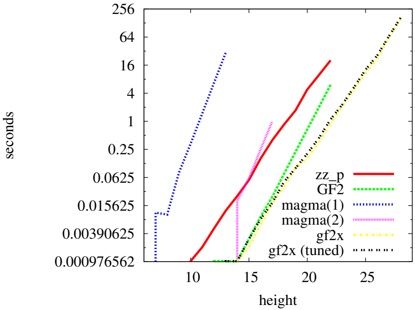
Construction of the tower + precomputations

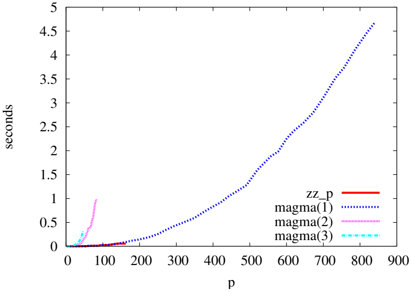


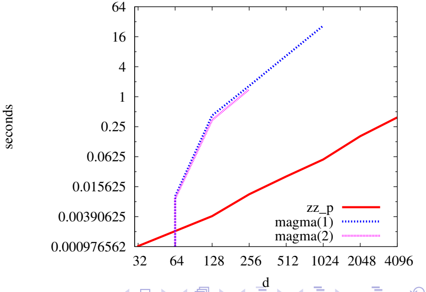




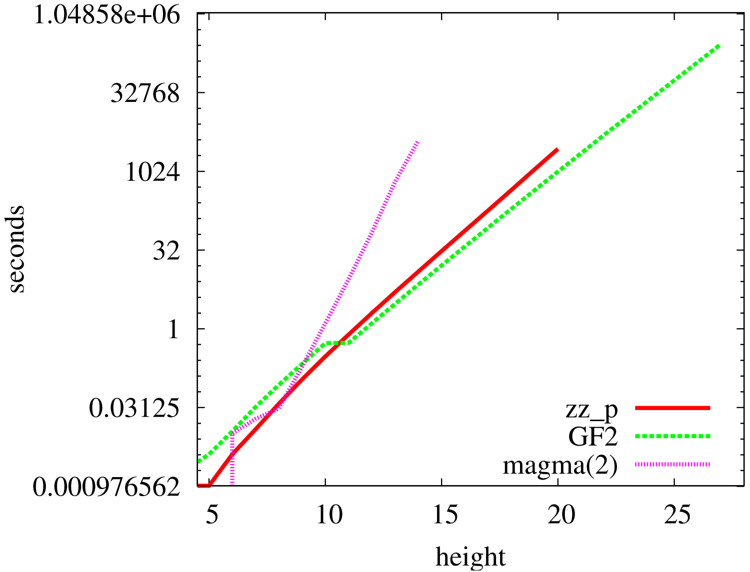
Multiplication





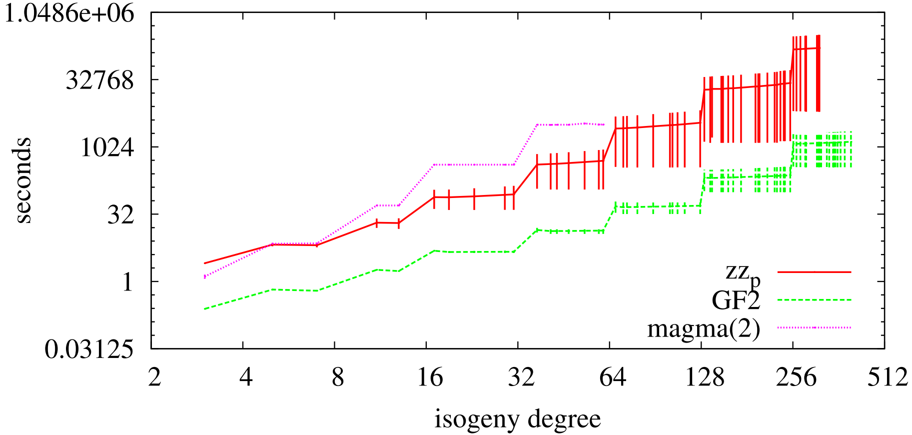


Isomorphism ([Couveignes '00] vs Magma)



Benchmarks on isogenies ([Couveignes '96])

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



FAAST

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.