# Fast arithmetics for Artin-Schreier extensions


| | |
|---|---|
| **Luca De Feo** | **Éric Schost** |
| **Tanc project** | **ORCCA** |
| **École polytechnique** | **UWO** |

# Artin-Schreier

## Artin-Schreier polynomials

- if $\mathbb{K}$ a finite field of characteristic $p$ and $\alpha$ is in $\mathbb{K}$,

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

  is an Artin-Schreier polynomial.

## Artin-Schreier extensions

- if $P$ an irreducible Artin-Schreier polynomial and

$$\mathbb{L} = \mathbb{K}[X]/P,$$

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

# Artin-Schreier towers

Starting from $\mathbb{U}_0 = \mathbb{F}_p[X_0]/Q(X_0)$ $\deg(Q) = d$

- take $a_1$ in $\mathbb{U}_0$ and let

$$P_1 = X_1^p - X_1 - a_1, \quad \mathbb{U}_1 = \mathbb{U}_0[X_1]/P_1$$

- take $a_2$ in $\mathbb{U}_1$ and let

$$P_2 = X_2^p - X_2 - a_2, \quad \mathbb{U}_2 = \mathbb{U}_1[X_2]/P_2$$

- continuing up to $k$, we get the tower $(\mathbb{U}_0, \dots, \mathbb{U}_k)$.

# Artin-Schreier towers

Starting from $\mathbb{U}_0 = \mathbb{F}_p[X_0]/Q(X_0)$ $\deg(Q) = d$

- take $a_1$ in $\mathbb{U}_0$ and let

$$P_1 = X_1^p - X_1 - a_1, \quad \mathbb{U}_1 = \mathbb{U}_0[X_1]/P_1$$

- take $a_2$ in $\mathbb{U}_1$ and let

$$P_2 = X_2^p - X_2 - a_2, \quad \mathbb{U}_2 = \mathbb{U}_1[X_2]/P_2$$

- continuing up to $k$, we get the tower $(\mathbb{U}_0, \ldots, \mathbb{U}_k)$.

Needed

- basic algorithms
- culminating with an isomorphism algorithm (Couveignes 00)

Motivation

- $p$-torsion points of elliptic curves,
- for isogeny computation (Couveignes 96).

# Complexity issues

Questions:

- $\mathbb{F}_p$-basis for $\mathbb{U}_i$?
- cost of arithmetic operations in this basis?
- cost of going up and down between $\mathbb{U}_i$ and $\mathbb{U}_{i+1}$?

Size of the problem

- $\dim_{\mathbb{F}_p}(\mathbb{U}_i) = \delta_i$, with $\delta_i = p^i d$
- we want algorithms of cost linear in $\delta_i$.

We almost achieve this, using the ideas of (Couveignes 00).

# Choosing a basis

# Multivariate basis

One can see $\mathbb{U}_i$ as

$$\mathbb{U}_i = \mathbb{F}_p[X_0, \ldots, X_i]/I,$$

with $I$ generated by

$$\left|\begin{array}{l} P_i = X_i^p - X_i - A_{i-1}(X_0, \ldots, X_{i-1}) \\ \quad\vdots \\ P_1 = X_1^p - X_1 - A_0(X_0) \\ Q_0(X_0) \end{array}\right.$$

Consequence: multivariate basis for $\mathbb{U}_i$

$$\{X_0^{e_0} X_1^{e_1} \cdots X_i^{e_i} \mid e_0 < d,\ e_1 < p,\ \ldots,\ e_i < p\}.$$

# Pros and cons

Pros: going up /down is easy

- insert zeros / remove zeros

Cons: multiplication is slow

- direct approach: expand and reduce
  - after expansion, we have the monomials

$$\{X_0^{e_0} X_1^{e_1} \cdots X_i^{e_i} \;\mid e_0 < 2d-1,\ e_1 < 2p-1,\ \ldots,\ e_i < 2p-1\}$$

  - so roughly $d(2p-1)^i$ coefficients, e.g. $p = 2$, $d = 1$: $2^i \to 3^i$
  - not linear in $\delta_i$
- indirect approach (Bostan *et al.*)
  - homotopy techniques and evaluation / interpolation
  - successful on some patterns
  - but not on this one: the cost is $d(2p-1)^i$ as well

# Univariate bases

At level $i$

- if we find a generator $y_i$ of $\mathbb{U}_i/\mathbb{F}_p$
- then $1, y_i, \ldots, y_i^{\delta_i - 1}$ is a basis of $\mathbb{U}_i$

Pros: arithmetic is fast

- let $\mathsf{M}(n)$ be a multiplication time

  Schönhage-Strassen's FFT $\quad \mathsf{M}(n) \in O(n \log(n) \log\log(n))$
- multiplication in $\mathbb{U}_i$ $\quad O(\mathsf{M}(\delta_i))$
- inversion in $\mathbb{U}_i$ $\quad O(\mathsf{M}(\delta_i) \log(\delta_i))$

Cons:

- going up /down is not obvious anymore

# Primitive towers

# Primitive towers

Primitive tower

- a tower is primitive if $\mathbb{U}_i = \mathbb{F}_p[x_i]$
- in this case, $Q_i$ is its minimal polynomial over $\mathbb{F}_p$ $\deg(Q_i) = \delta_i$

Remark: not always the case

- $P_1 = X_1^p - X_1 - 1$.

Theorem (extends a result in (Cantor '89))
If $\mathrm{Tr}_{\mathbb{U}_0/\mathbb{F}_p}(x_0) \neq 0$, the tower defined by

$$\left| \begin{array}{lcll} P_1 & = & X_1^p - X_1 - X_0 & \\ P_i & = & X_i^p - X_i - X_{i-1}^{2p-1} & i > 1 \end{array} \right.$$

is primitive.

From now on, we work in this specific tower

## Setup: finding $Q_i$

Algorithm essentially in (Cantor '89)

### Low levels

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

### Higher levels: $\omega$ is a $2p-1$-th root of unity

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

### Cost

- $O\left(\mathsf{M}(p^{i+1}d)\log p\right)$
- Up to logs, this is $O(p^{i+1}d)$

# Univariate and bivariate

## Univariate basis

- the basis $1, x_i, \ldots, x_i^{\delta_i - 1}$
- computations done modulo $Q_i(X_i)$
- $v \dashv \mathbb{U}_i$ indicates that $v$ is written on this basis

## Bivariate basis

- if we see $\mathbb{U}_i$ as $\mathbb{U}_{i-1}[X_i]/P_i$, any $v$ in $\mathbb{U}_i$ can be written as

$$v = v_0(x_{i-1}) + \cdots + v_{p-1}(x_{i-1}) x_i^{p-1},$$

  with $v_i \dashv \mathbb{U}_{i-1}$.

- computations done modulo

$$\left| \begin{array}{l} P_i(X_{i-1}, X_i) = X_i^p - X_i - X_{i-1}^{2p-1} \\ Q_{i-1}(X_{i-1}) \end{array} \right.$$

# Push-down and Lift-up

## 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 $v = v_0 + \cdots + v_{p-1} x_i^{p-1}$

## Theorem

- Both operations can be done in time

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

- Up to logs, this is $O(p^{i+1} d)$.

# Easy direction: push-down

We want to reduce $v(X_i)$ modulo

$$\left| \begin{array}{l} P_i(X_{i-1}, X_i) = X_i^p - X_i - X_{i-1}^{2p-1} \\ Q_{i-1}(X_{i-1}) \end{array} \right.$$

Algorithm: reduction modulo $P_i$

Example with $p = 2$, we work modulo $X_i^2 - X_i - X_{i-1}^3$

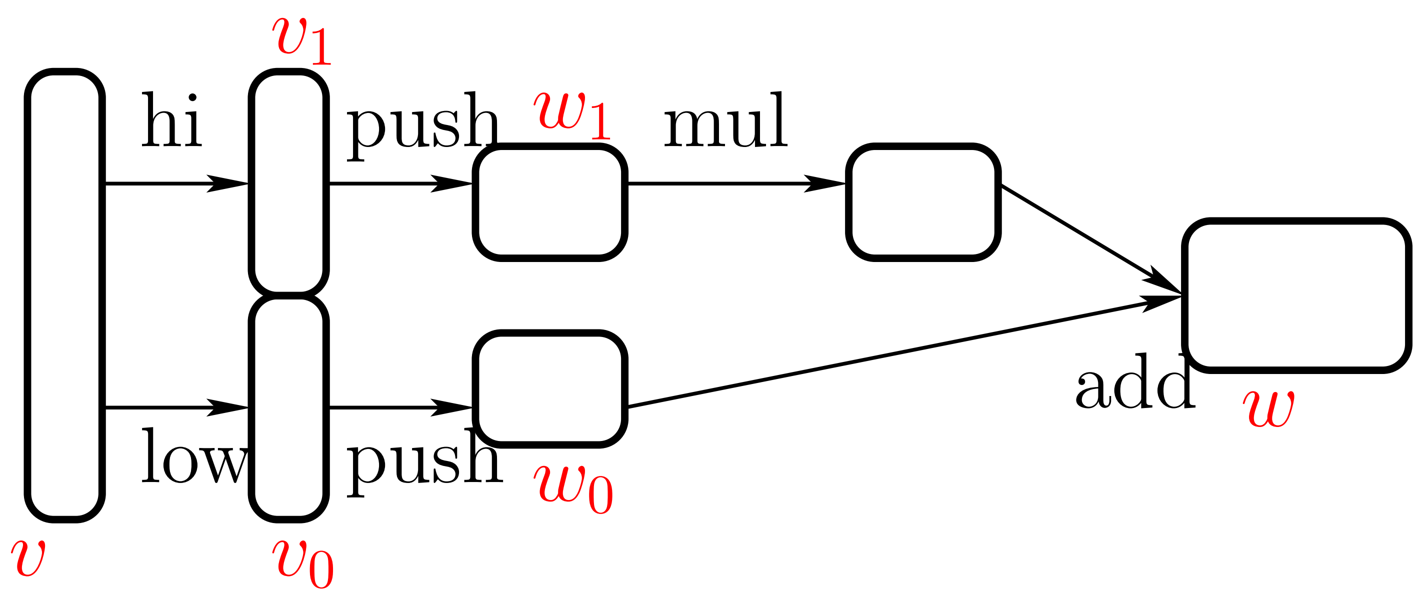
- assume $\deg(v) < 2^n$
- write $v = v_0(X_i) + X_i^{2^{n-1}} v_1(X_i)$
- process recursively $v_0$ and $v_1$, getting $w_0$ and $w_1$
- remark that $X_i^{2^{n-1}} = X_i + X_{i-1}^3 + X_{i-1}^6 + \cdots + X_{i-1}^{3 \cdot 2^{n-2}} \bmod P_i$
- return $w_0 + (X_i + X_{i-1}^3 + X_{i-1}^6 + \cdots + X_{i-1}^{3 \cdot 2^{n-2}}) w_1 \bmod P_i$

# Easy direction: push-down

We want to reduce $v(X_i)$ modulo

$$\left| \begin{array}{l} P_i(X_{i-1}, X_i) = X_i^p - X_i - X_{i-1}^{2p-1} \\ Q_{i-1}(X_{i-1}) \end{array} \right.$$

Algorithm: reduction modulo $P_i$



# Harder direction: lift-up

# Using trace formulas

Given $w(X_{i-1}, X_i)$, we want to find $v(X_i)$ such that $w = v$ modulo

$$\left|\begin{array}{l} P_i(X_{i-1}, X_i) = X_i^p - X_i - X_{i-1}^{2p-1} \\ Q_{i-1}(X_{i-1}) \end{array}\right.$$

Trace formulas: (Rouillier 99)

- let $\mathrm{Tr}'$ be the linear form $a \mapsto \mathrm{Tr}(aw)$,
- then given the values of $\mathrm{Tr}'$ on the univariate basis $1, x_i, \ldots, x_i^{\delta_i - 1}$,
- one can recover $v$ using a few more operations.

How: the generating series

$$\sum_{j \geqslant 0} \mathrm{Tr}'(x_i^j) X_i^j$$

is rational; its denominator is (essentially) $Q_i$ and its numerator is (essentially) $v$.

# Duality

Multiplication-by-$w$

- from the bivariate basis to itself

Transposed multiplication-by-$w$

- From the dual-bivariate basis to itself
- concretely:
  - **input:** the values of a linear form $\ell$ on the bivariate basis,
  - **output:** the values of $\ell'$ on the univariate basis, with $\ell' : a \mapsto \ell(aw)$.

Starting from Tr, this gives us the values of $\mathrm{Tr}'$ on the bivariate basis.

# Duality, cont.

## Push-down

- change-of-basis from univariate to bivariate

## Transposed push-down

- change-of-basis from dual-bivariate to dual-univariate
- concretely:
  - **input:** the values of a linear form $\ell$ on the bivariate basis,
  - **output:** the values of $\ell$ on the univariate basis.

Starting from $\mathrm{Tr}'$ on the bivariate basis, this gives us $\mathrm{Tr}'$ on the univariate basis.

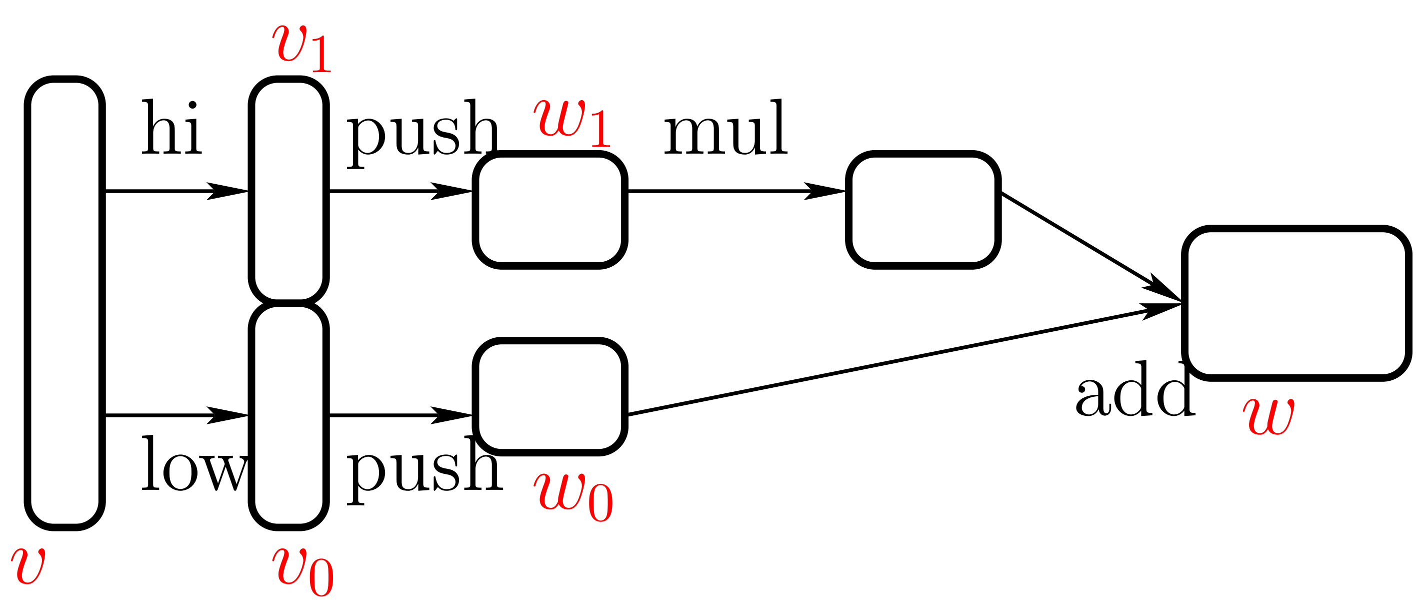
# Transposition principle

Given a linear algorithm computing a linear application, we can deduce another linear algorithm computing the transpose application in the same cost.

Fiduccia, Kaminski *et al.*, Shoup-Kaltofen, . . .

Reverse the "flow" of the program

- order of the iterations are reversed
- basic subroutine: transposed multiplication



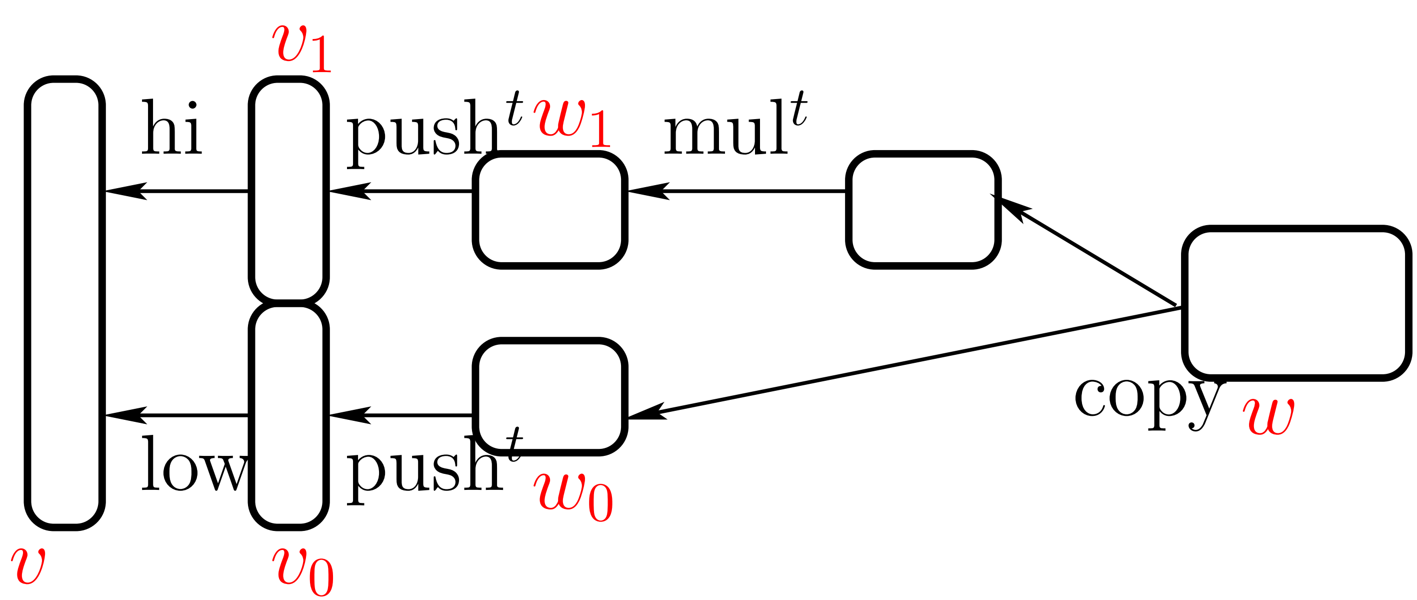
# Transposition principle

Given a linear algorithm computing a linear application, we can deduce another linear algorithm computing the transpose application in the same cost.

Fiduccia, Kaminski *et al.*, Shoup-Kaltofen, . . .

Reverse the "flow" of the program

- order of the iterations are reversed
- basic subroutine: transposed multiplication



# Experiments and applications

# Speeding up more operations

## Divide and conquer

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

## Where it works (Couveignes 00)

- trace, $p$-th roots, inverse, iterated Frobenius, ...
- isomorphism

## Theorem

- One can apply an isomorphism (and its inverse) between any Artin-Schreier towers of height $i$ in time $O(p^{i+1}d)$ (up to logs).

# Example: iterated Frobenius

Wanted: $v \mapsto v^{p^{p^j d}}$

- $v \in \mathbb{U}_j \;\Rightarrow\; v^{p^{p^j d}} = v$,
- $x_i^{p^{p^j d}} = x_i + \beta_{i-1,j}$ where $\beta_{i-1,j} = \sum_{h=0}^{p^j d-1} (x_{i-1}^{2p-1})^{p^h}$,
- $v^{p^{p^j d}} = \sum_{h=0}^{p-1} v_h^{p^{p^j d}} (x_i + \beta_{i-1,j})^h$

IterFrobenius

**Input:** $v$, $i$, $j$ with $v \dashv \mathbb{U}_i$ and $j \geqslant 0$.

**Output:** $v^{p^{p^j d}} \dashv \mathbb{U}_i$.

- If $i \leq j$, return $v$
- Let $v_0 + v_1 x_i + \cdots + v_{p-1} x_i^{p-1} = \mathsf{Push\text{-}down}(v)$,
- for $h \in [0, \ldots, p-1]$, let $t_h = \mathsf{IterFrobenius}(v_h, i-1, j)$
- let $w = \sum_{h=0}^{p-1} t_h (x_i + \beta_{i-1,j})^h$
- return $\mathsf{Lift\text{-}up}(w)$

# Implementation

## Implementation in NTL

- `GF2`: $p = 2$, FFT, bit optimisations (gf2x Brent et al.)
- `zz_p`: $p < 2^{53}$, FFT, no bit-tricks,
- `ZZ_p`: generic $p$, like `zz_p` but slower.

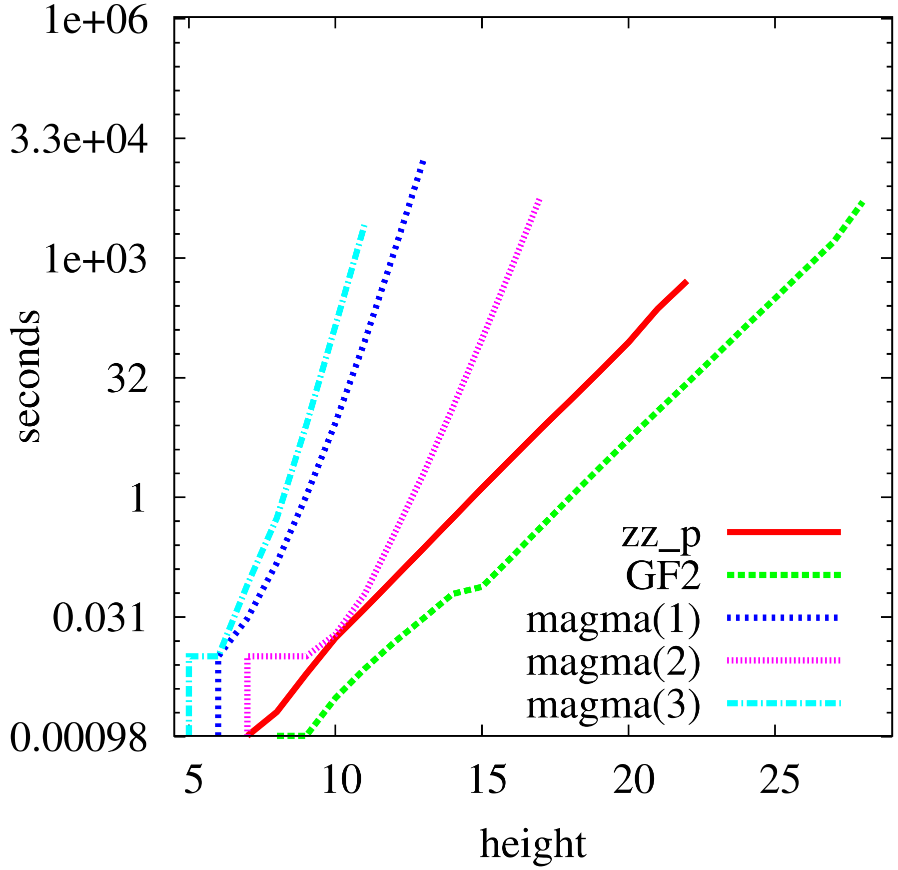
## Comparison to Magma

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

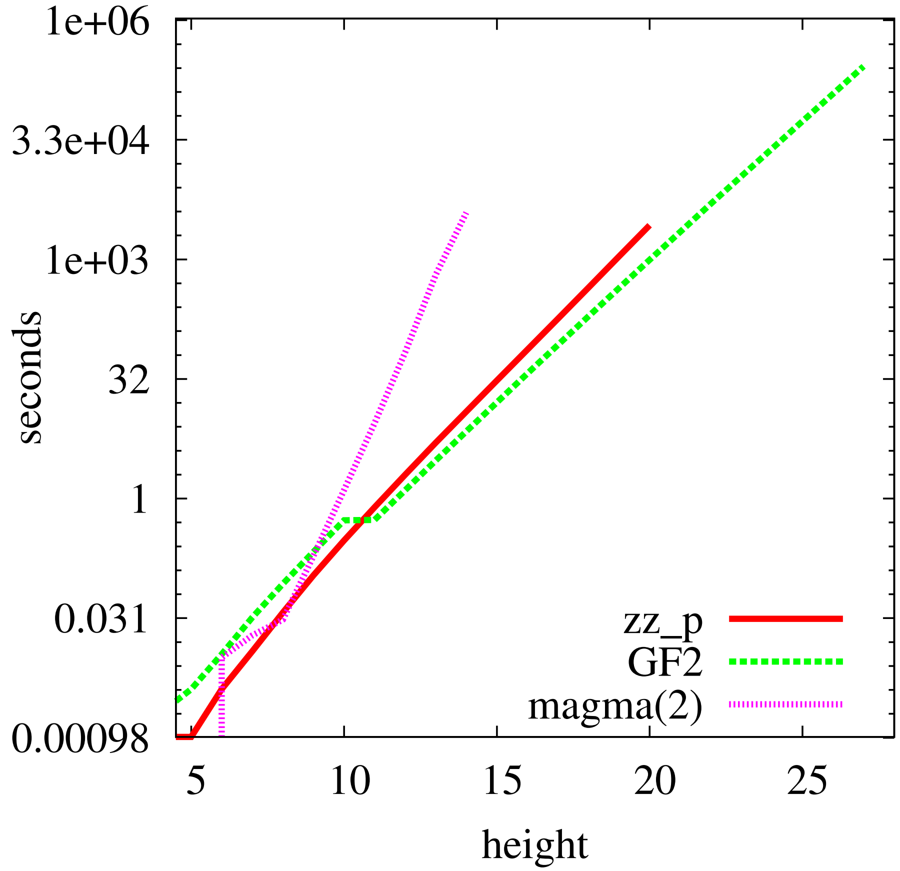
## Benchmarks (AMD Opteron 2500)

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

# Benchmark: building the tower



# Benchmark: constructing an isomorphism



# Couveignes' isogeny algorithm

## In a nutshell

- to find an $\ell$-isogeny between elliptic curves $\mathscr{E}$ and $\mathscr{E}'$
- build $p^k$-torsion for $\mathscr{E}$ and $\mathscr{E}'$
  - two Artin-Schreier towers
  - $p^k \simeq \ell$
- set-up an isomorphism between them
- find the isogeny by interpolation, by trial-and-error

Improvements by De Feo.

# Benchmark: isogenies