

# Computing isogenies of elliptic curves in small characteristics

L. De Feo

Projet TANC, LIX, École Polytechnique, France

Journées Arithmétiques

July 9, 2009

Université Jean Monnet - St. Étienne

# Compute isogenies

## What?

(Separable) isogenies: (separable) non-constant regular maps of elliptic curves that are group homomorphism

- Finite kernel, onto, given by one rational fractions,
- we are specially interested elliptic curves over finite fields.

## Why?

- Point counting.
- Proving hardness of discrete logarithm.
- Move discrete logarithms to easier curves.
- Speeding up point multiplication.
- Hide weak curves behind chains of isogenies.
- Define hash functions.

## Separable isogenies, odd degree (simplified Weierstrass model)

$$\mathcal{I}(X, Y) = \left( \frac{g(X)}{h^2(X)}, cY \left( \frac{g(X)}{h^2(X)} \right)' \right)$$

$$\ell = \deg \mathcal{I} = \# \ker \mathcal{I} = 2 \deg h + 1 \quad \text{odd.}$$

# Compute isogenies

## What?

(Separable) isogenies: (separable) non-constant regular maps of elliptic curves that are group homomorphism

- Finite kernel, onto, given by one rational fractions,
- we are specially interested elliptic curves over finite fields.

## Why?

- Point counting.
- Proving hardness of discrete logarithm.
- Move discrete logarithms to easier curves.
- Speeding up point multiplication.
- Hide weak curves behind chains of isogenies.
- Define hash functions.

## Normalised (or strict) isogenies

$$\mathcal{I}(X, Y) = \left( \frac{g(X)}{h^2(X)}, c_Y \left( \frac{g(X)}{h^2(X)} \right)' \right)$$

$$\ell = \deg \mathcal{I} = \# \ker \mathcal{I} = 2 \deg h + 1 \quad \text{odd.}$$

# Compute isogenies

## What?

(Separable) isogenies: (separable) non-constant regular maps of elliptic curves that are group homomorphism

- Finite kernel, onto, given by one rational fractions,
- we are specially interested elliptic curves over finite fields.

## Why?

- Point counting.
- Proving hardness of discrete logarithm.
- Move discrete logarithms to easier curves.
- Speeding up point multiplication.
- Hide weak curves behind chains of isogenies.
- Define hash functions.

## Normalised (or strict) isogenies

$$\mathcal{I}(X, Y) = \left( \frac{g(X)}{h^2(X)}, Y \left( \frac{g(X)}{h^2(X)} \right)' \right)$$

$$\ell = \deg \mathcal{I} = \# \ker \mathcal{I} = 2 \deg h + 1 \quad \text{odd.}$$

# Vélu formula

## Vélu formula for algebraically closed fields

$$E : y^2 = x^3 + ax + b$$

Being given the points of a subgroup  $H$  of  $E$ ,

$$\mathcal{I}(\mathcal{O}_E) = \mathcal{I}(\mathcal{O}_{E/H})$$

$$\mathcal{I}(P) = \left( x(P) + \sum_{Q \in H - \{\mathcal{O}_E\}} x(P+Q) - x(Q) \ , \right. \\ \left. y(P) + \sum_{Q \in H - \{\mathcal{O}_E\}} y(P+Q) - y(Q) \right).$$

The curve  $E' = E/C$  is given by simple formulae. This is the normalised isogeny.

## Rational isogenies on generic fields

Knowing  $h^2(X) = \prod_{Q \in H - \{\mathcal{O}_E\}} (X - x(Q))$  is enough.

# Computing isogenies: which problem?

## $j$ -invariant

$$E : y^2 = x^3 + ax + b \qquad j(E) = \frac{1728(4a)^3}{16(4a^3 + 27b^2)}$$

## Modular polynomial

$$\Phi_\ell(j(E), j(E')) = 0 \quad \text{iff } E \text{ } \ell\text{-isogenous to } E'$$

- Symmetric polynomial, degree  $\ell$ , integer coefficients of  $\tilde{O}(\ell)$  bits.
- Computed in  $\tilde{O}(\ell^3)$  bit operations.

## Which problem?

- 1 Given  $E$ , find an  $\ell$ -isogenous curve and an  $\ell$ -isogeny.
  - 2 Given  $E$  and  $E'$ , find an  $\ell$ -isogeny.
- Traditional solution to 1: find a curve by factoring  $\Phi_\ell(X, j(E))$ , then solve 2.
  - In SEA one needs 1, other applications require 2. We'll focus on 2.

# Computing isogenies in $\mathbb{C}$

## Elliptic functions

$$E \cong \mathbb{C}/(\omega_1\mathbb{Z} + \omega_2\mathbb{Z}) \xrightarrow{\mathcal{I}} \mathbb{C}/\left(\frac{\omega_1}{\ell}\mathbb{Z} + \omega_2\mathbb{Z}\right) \cong E'$$
$$z \longmapsto z$$

## Weierstrass functions

$$\wp_E(z) = z^{-2} + \sum_{k=1}^{\infty} c_k z^{2k} \quad \text{with}$$

$$c_1 = -\frac{a}{5}, \quad c_2 = -\frac{b}{7}, \quad c_k = \frac{3}{(k-2)(2k+3)} \sum_{j=1}^{k-2} c_j c_{k-1-j}$$

and they verify

$$\left\{ \begin{array}{l} \wp_E'^2 = 4\wp_E^3 + 4a\wp_E + 4b, \\ \wp_{E'}(z) = \sum_{i=0}^{\ell-1} \wp_E\left(z + i\frac{\omega_1}{\ell}\right) - \wp_E\left(i\frac{\omega_1}{\ell}\right). \end{array} \right.$$

# The large characteristic case

## Weierstrass functions

$$\wp_E(z) = z^{-2} + \sum_{k=1}^{\infty} c_k z^{2k} \quad \text{with}$$

$$c_1 = -\frac{a}{5}, \quad c_2 = -\frac{b}{7}, \quad c_k = \frac{3}{(k-2)(2k+3)} \sum_{j=1}^{k-2} c_j c_{k-1-j}$$

division by zero when  $2k + 3 \geq p$ .

## Large characteristic algorithms

Work with truncated power series with precision  $\ll \frac{p}{2}$ .

'91 Charlap, Coley, Robbins

$O(\ell^2)$

'92 Elkies

$\tilde{O}(\ell^2)$

'92 Atkin

$\tilde{O}(\ell^2)$

'98 Elkies

$\tilde{O}(\ell^2)$

'08 Bostan, Morain, Salvy, Schost

$\tilde{O}(\ell)$

# The large characteristic case

## Weierstrass functions

$$\wp_E(z) = z^{-2} + \sum_{k=1}^{\infty} c_k z^{2k} \quad \text{with}$$

$$c_1 = -\frac{a}{5}, \quad c_2 = -\frac{b}{7}, \quad c_k = \frac{3}{(k-2)(2k+3)} \sum_{j=1}^{k-2} c_j c_{k-1-j}$$

division by zero when  $2k + 3 \geq p$ .

## Large characteristic algorithms **\*Only work for normalised isogenies**

Work with truncated power series with precision  $\ll \frac{p}{2}$ .

'91 Charlap, Coley, Robbins\*

$O(\ell^2)$

'92 Elkies

$\tilde{O}(\ell^2)$

'92 Atkin

$\tilde{O}(\ell^2)$

'98 Elkies

$\tilde{O}(\ell^2)$

'08 Bostan, Morain, Salvy, Schost\*

$\tilde{O}(\ell)$

# Using $p$ -adics: Lercier-Sirvent

## Weierstrass functions

$$\wp_E(z) = z^{-2} + \sum_{k=1}^{\infty} c_k z^{2k} \quad \text{with}$$

$$c_1 = -\frac{a}{5}, \quad c_2 = -\frac{b}{7}, \quad c_k = \frac{3}{(k-2)(2k+3)} \sum_{j=1}^{k-2} c_j c_{k-1-j}$$

## Lercier, Sirvent 2009

Work in the  $p$ -adics to avoid divisions by zero.

- Lift  $E$  to  $\bar{E}$  in  $\mathbb{Q}_q$ .
- Problem: the lift of  $E'$  is not necessarily normalised.
- Lift  $\Phi_\ell$ , factor  $\bar{\Phi}_\ell(X, j(\bar{E}))$  to obtain a normalised  $\bar{E}'$ ,
- use BMMS to compute the lifted isogeny, then reduce.

Works for any  $p$ , complexity  $\tilde{O}(\ell^3)$ , but solves problem 1 directly.

# Using the $p$ -torsion

## Other algorithms

'94 Couveignes I

$O(\ell^3)$

'96  $p = 2$ , Lercier

$\Omega(\ell^3)$  ?

'96 Couveignes II (+ D.F.)

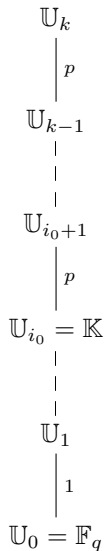
$\tilde{O}(\ell^2)$

## Couveignes II

Interpolate a polynomial on a large enough subgroup of  $E$ , reconstruct the isogeny from there.

- $\mathcal{I}(E[p^k]) = E'[p^k]$ ,  $E[p^k] \cong E'[p^k] \cong \mathbb{Z}/p^k\mathbb{Z}$ ,
- for  $E[p^k] = \langle P \rangle$  there's  $\phi(p^k)$  possible images of  $P$  in  $E'[p^k]$ ,
- try them all until a fraction of the form  $\frac{g(X)}{h^2(X)}$  is found.
- One needs  $\phi(p^k) > 4\ell$  in order to succeed.

# Structure of the $p^k$ -torsion



## Definition ( $p^k$ -torsion tower)

$(\mathbb{F}_q = \mathbb{U}_0, \dots, \mathbb{U}_k)$  is the tower of field extensions of minimal degree s.t. for any  $i$

$$E[p^i] \subset E(\mathbb{U}_i).$$

## Theorem (Structure of $(\mathbb{U}_0, \dots, \mathbb{U}_k)$ )

There is a  $i_0 \geq 1$  s.t.  $\mathbb{U}_{i_0} = \mathbb{U}_1$  and for  $i \geq i_0$

$$[\mathbb{U}_{i+1} : \mathbb{U}_i] = p.$$

And  $[\mathbb{U}_1 : \mathbb{U}_0]$  divides  $p - 1$ .

## Complexity

In general towers tend to explode early. In practice  $[\mathbb{U}_k : \mathbb{F}_q] \sim p^k$

- Interpolation of degree  $p^k$  in  $\mathbb{U}_k$  costs  $\tilde{O}(p^k)$  operations in  $\mathbb{U}_k$ ,
- that is  $\tilde{O}(p^{2k} \log q)$  operations in  $\mathbb{F}_p$ , to be repeated  $O(\phi(p^k))$  times. Total complexity is  $\tilde{O}(\ell^3)$ .

# Faster interpolation using effective Galois groups

Interpolation of  $v_i \mapsto s_i$  is defined modulo  $T$ , where

$$T(X) = \prod_i (X - v_i)$$

## Lagrange formula

$$A(X) = \sum_{i=1}^n v_i \frac{T(X)}{X - v_i} \prod_{j \neq i} \frac{1}{v_i - v_j}$$

# Faster interpolation using effective Galois groups

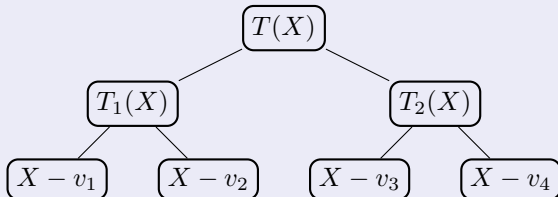
Interpolation of  $v_i \mapsto s_i$  is defined modulo  $T$ , where

$$T(X) = \prod_i (X - v_i)$$

## Lagrange formula

$$A(X) = \sum_{i=1}^n \frac{s_i}{T'(v_i)} \cdot \frac{T(X)}{X - v_i}$$

## Interpolation by chinese remaindering



# Faster interpolation using effective Galois groups

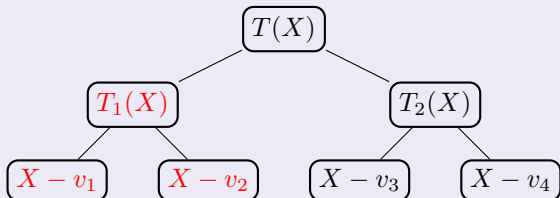
Interpolation of  $v_i \mapsto s_i$  is defined modulo  $T$ , where

$$T(X) = \prod_i (X - v_i)$$

## Lagrange formula

$$A_1(X) = \frac{s_1}{T'(v_1)(X - v_2)} + \frac{s_2}{T'(v_2)(X - v_1)}$$

## Interpolation by chinese remaindering



# Faster interpolation using effective Galois groups

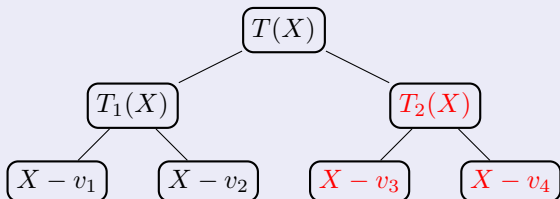
Interpolation of  $v_i \mapsto s_i$  is defined modulo  $T$ , where

$$T(X) = \prod_i (X - v_i)$$

## Lagrange formula

$$A_2(X) = \frac{s_3}{T'(v_3)(X - v_4)} + \frac{s_4}{T'(v_4)(X - v_3)}$$

## Interpolation by chinese remaindering



# Faster interpolation using effective Galois groups

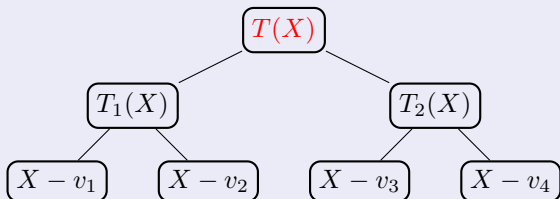
Interpolation of  $v_i \mapsto s_i$  is defined modulo  $T$ , where

$$T(X) = \prod_i (X - v_i)$$

## Lagrange formula

$$A(X) = T_2(X)A_1(X) + T_1(X)A_2(X)$$

## Interpolation by chinese remaindering



# Faster interpolation using effective Galois groups

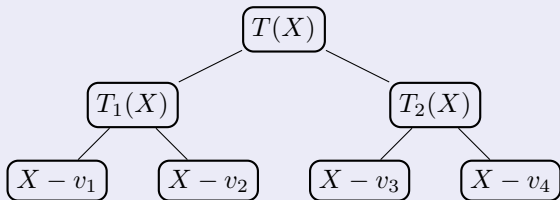
Interpolation of  $v_i \mapsto s_i$  is defined modulo  $T$ , where

$$T(X) = \prod_i (X - v_i)$$

## Lagrange formula

Complexity  $\tilde{O}(n)$  operations in the coefficient ring.

## Interpolation by chinese remaindering



# Faster interpolation using effective Galois groups

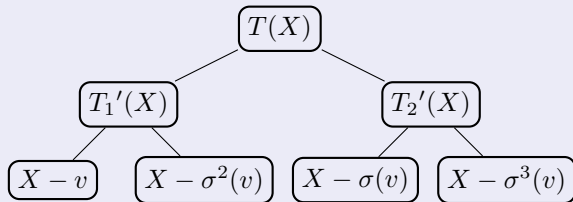
Interpolation of  $v_i \mapsto s_i$  is defined modulo  $T$ , where

$$T(X) = \prod_i (X - v_i)$$

## Lagrange formula

Let now  $v \in \mathbb{U}_2$ ,  $\sigma \in \text{Gal}(\mathbb{U}_2/\mathbb{U}_0)$  and  $v_i = \sigma^{i-1}(v)$ . Rearrange the tree, then  $T'_1, T'_2 \in \mathbb{U}_1[X]$  and  $T \in \mathbb{U}_0[X]$ .

## Interpolation by chinese remaindering



# Faster interpolation using effective Galois groups

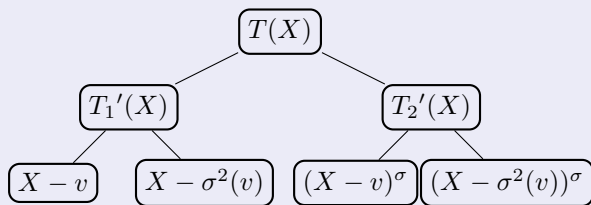
Interpolation of  $v_i \mapsto s_i$  is defined modulo  $T$ , where

$$T(X) = \prod_i (X - v_i)$$

## Lagrange formula

For a polynomial  $P$  note  $P^\sigma$  the action on the coefficients of  $P$ , then

## Interpolation by chinese remaindering



# Faster interpolation using effective Galois groups

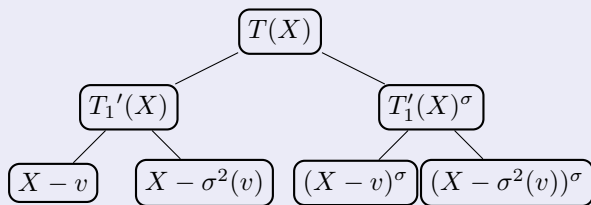
Interpolation of  $v_i \mapsto s_i$  is defined modulo  $T$ , where

$$T(X) = \prod_i (X - v_i)$$

## Lagrange formula

For a polynomial  $P$  note  $P^\sigma$  the action on the coefficients of  $P$ , then

## Interpolation by chinese remaindering



# Faster interpolation using effective Galois groups

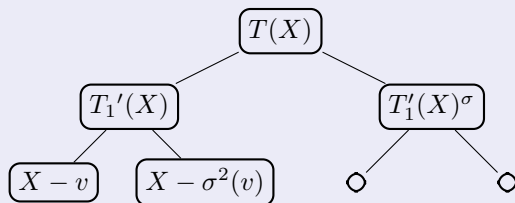
Interpolation of  $v_i \mapsto s_i$  is defined modulo  $T$ , where

$$T(X) = \prod_i (X - v_i)$$

## Lagrange formula

Complexity  $\tilde{O}(n)$  operations in  $\mathbb{U}_0$ .

## Interpolation by chinese remaindering



# Reducing the number of interpolations

## Couveignes' algorithm

- We need  $O(p^k)$  interpolations, each sending  $P \in E[p^k]$  over  $Q \in E'[p^k]$ ,
- $Q, R \in E'[p^k]$  are conjugates tied by the relation  $Q = \varphi_q^i(P)$  for some  $i$ .

## Using modular composition

Let  $A_Q$  be the polynomial with coefficients in  $\mathbb{F}_q$  sending  $P$  over  $Q$ , then

$$A_Q([j]P) = [j]Q \text{ for every } j.$$

Now let  $\varphi_q(Q) = [\lambda]Q$ , then  $A_Q(\varphi_q([j]P)) = [j][\lambda]Q$ .

So  $A_Q \circ \varphi_q = A_{[\lambda]Q} \bmod T$ . Solving this is *modular composition*.

## Modular composition

- Theoretical complexity  $O(\ell \log_p q)$ , practical complexity  $O(\ell^2 \log_p^2 q) \dots$
- $\dots$  but still much faster than a single interpolation.

# Effective Artin-Schreier towers (D.F., Schost '09)



## Primitive towers

- Find special *primitive* Artin-Schreier towers s.t. the generator of  $\mathbb{U}_i/\mathbb{U}_{i-1}$  is also a generator of  $\mathbb{U}_i/\mathbb{F}_p$ .
- Use univariate representation over  $\mathbb{F}_p$  to perform fast arithmetics (FFT multiplication, Newton inversion, etc.).
- Use fast isomorphisms to treat generic towers of extensions.

## Level embedding

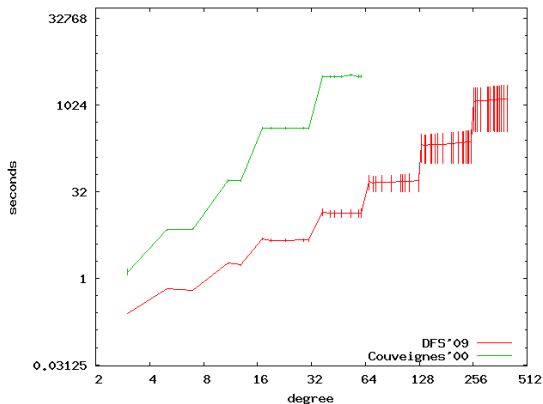
- Move up and down in the structure,
- Effective frobenius maps.

# Implementation

- Implementation in C++ with NTL.
- Porting code to SAGE.
- Some parts of the code are distributed already, if you can't wait:  
<http://www.lix.polytechnique.fr/Labo/Luca.De-Feo/FAAST>

# Benchmarks

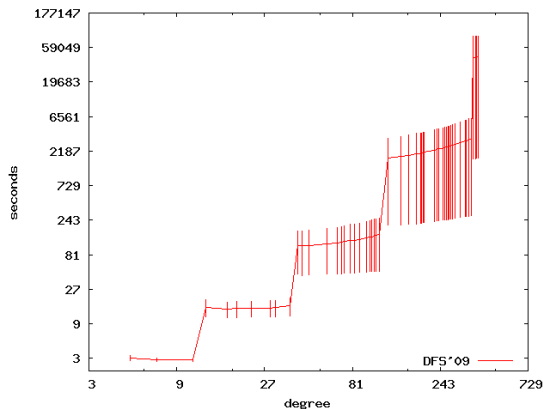
- NTL implementation vs. Magma implementation (no fast Galois groups)
- $\mathbb{K} = \mathbb{F}_{2^{101}}$ .



$\ell$	$E[p^k]$	$E'[p^k]$	Interp	Step 6	ModComp	Avg tries	Avg loop time
31	1.3128	1.3128	1.1058	0.00218	0.00218	64	0.279
61	3.5454	3.5464	2.5236	0.00783	0.00900	128	2.154
127	9.2975	9.3026	5.6881	0.03147	0.03634	256	17.359
251	23.7984	23.7984	12.7251	0.12415	0.14519	512	137.902
397	59.7439	59.7579	28.3387	0.36822	0.58027	1024	971.254

# Benchmarks

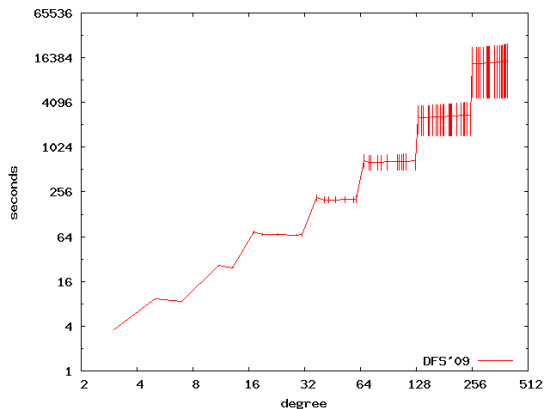
- NTL implementation
- $\mathbb{K} = \mathbb{F}_{364}$ .



$\ell$	$E[p^k]$	$E'[p^k]$	Interp	Step 6	ModComp	Avg tries	Avg loop time
11	0.6109	0.6109	0.4669	0.0194	0.0249	13	0.58
37	2.3946	2.3916	2.1066	0.1988	0.1381	40	13.48
113	9.8045	9.8055	8.5377	1.7712	0.8690	121	319.47
359	38.3292	38.3972	34.7147	17.5004	7.0088	364	8921.35
389	159.8280	159.5690	147.741	45.1558	69.9133	1093	125770.52

# Record Timings!

- NTL implementation
- $\mathbb{K} = \mathbb{F}_{2^{1023}}$ .



$\ell$	$E[p^k]$	$E'[p^k]$	Interp	Step 6	ModComp	Avg tries	Avg loop time
31	21.182	21.174	11.597	0.0178	0.02541	64	2.768
61	58.656	58.665	26.826	0.0645	0.10398	128	21.576
127	154.357	154.296	61.202	0.2580	0.41578	256	172.503
251	383.773	383.861	138.428	0.9950	1.66120	512	1360.000
397	931.022	931.610	313.609	3.1819	6.73608	1024	10156.011