Fast algorithms: from type theory to number theory

Luca De Feo

INRIA Saclay, Projet TANC

October 25, 2010
Séminaire Algorithmes
INRIA Rocquencourt, Le Chesnay
Elliptic curve cryptography

Weierstrass form: \( y^2 = x^3 + ax + b \);

Group law: Chord-tangent;

Crypto: Based on discrete log in \( E(\mathbb{F}_q) \);

Hasse bound: \(|\#E(\mathbb{F}_q) - q - 1| \leq 2\sqrt{q}\).
Isogenies are group morphisms of elliptic curves:

\[ J : E \rightarrow E' \]

\[ J(x, y) = \left( \frac{g(x)}{h(x)}, cy \left( \frac{g(x)}{h(x)} \right)' \right) \]

What do you do with an isogeny over a finite field?

- Point counting (Schoof 1995);
- Speed up point multiplication (Gallant, Lambert, and Vanstone 2001);
- Reduce a Discrete Logarithm Problem to another (Gaudry, Hess, and Smart 2002; Smith 2009);
- Construct new cryptosystems (Teske 2006; Rostovtsev and Stolbniov 2006);
- Construct hash functions (Charles, Lauter, and Goren 2009).
Isogenies: an example

The GHS attack (Gaudry, Hess, and Smart 2002)

\[ E / \mathbb{F}_{q^d} \]

- Given an elliptic curve \( E \) defined over a composite field \( \mathbb{F}_{q^d} \);
Isogenies: an example

The GHS attack (Gaudry, Hess, and Smart 2002)

\[ E / \mathbb{F}_{q^d} \xrightarrow{J} H / \mathbb{F}_q \]

- Given an elliptic curve \( E \) defined over a composite field \( \mathbb{F}_{q^d} \);
- Computes an isogeny to an hyperelliptic curve \( H \) defined over \( \mathbb{F}_q \).
- For certain parameters, the discrete log is easier on \( H \) than on \( E \).
Isogenies: an example

The GHS attack (Gaudry, Hess, and Smart 2002)

\[ E/F_{q^d} \xrightarrow{J} H/F_q \]

- Given an elliptic curve $E$ defined over a composite field $F_{q^d}$;
- Computes an isogeny to an hyperelliptic curve $H$ defined over $F_q$.
- For certain parameters, the discrete log is easier on $H$ than on $E$.

A trapdoor cryptosystem (Teske 2006)

**Fact:** Only a small fraction of the curves over $F_{q^d}$ is vulnerable to GHS

\[ E_{\text{trap}} \]

- Select a curve $E_{\text{trap}}$ vulnerable to GHS;
Isogenies: an example

The GHS attack (Gaudry, Hess, and Smart 2002)

\[ E/F_{q^d} \rightarrow^J H/F_q \]

- Given an elliptic curve \( E \) defined over a composite field \( F_{q^d} \);
- Computes an isogeny to an hyperelliptic curve \( H \) defined over \( F_q \);
- For certain parameters, the discrete log is easier on \( H \) than on \( E \).

A trapdoor cryptosystem (Teske 2006)

**Fact:** Only a small fraction of the curves over \( F_{q^d} \) is vulnerable to GHS

\[ E_{\text{trap}} \rightarrow E_{\text{pub}} \]

- Select a curve \( E_{\text{trap}} \) vulnerable to GHS;
- Take a random walk through the *isogeny graph*, land on a curve \( E_{\text{pub}} \) not vulnerable to GHS;
Isogenies: an example

The GHS attack (Gaudry, Hess, and Smart 2002)

Given an elliptic curve $E$ defined over a composite field $\mathbb{F}_{q^d}$;
Computes an isogeny to an hyperelliptic curve $H$ defined over $\mathbb{F}_q$.
For certain parameters, the discrete log is easier on $H$ than on $E$.

A trapdoor cryptosystem (Teske 2006)

Fact: Only a small fraction of the curves over $\mathbb{F}_{q^d}$ is vulnerable to GHS

Select a curve $E_{\text{trap}}$ vulnerable to GHS;
Take a random walk through the isogeny graph, land on a curve $E_{\text{pub}}$ not vulnerable to GHS;
Use $E_{\text{pub}}$ for public key cryptography, give $E_{\text{trap}}$ to a trusted authority for key escrow.
Isogenies: a challenge

Let

\[ \mathbb{F}_q = \mathbb{F}_2[Z]/(Z^{41} + Z^3 + 1) \]

The following two curves are isogenous:

\[ y^2 + xy = x^3 + 1/(Z^{36} + Z^{35} + Z^{34} + Z^{32} + Z^{31} + Z^{30} + Z^{26} + Z^{23} + Z^{22} + Z^{21} + Z^{20} + Z^{18} + Z^{17} + Z^{13} + Z^{12} + Z^{11} + Z^{8} + Z^{7} + Z^{5} + Z^{4} + Z^{2}) \]

\[ y^2 + xy = x^3 + 1/(Z^{40} + Z^{39} + Z^{38} + Z^{37} + Z^{35} + Z^{34} + Z^{28} + Z^{22} + Z^{15} + Z^{14} + Z^{11} + Z^{10} + Z^{9} + Z^{8} + Z^{7} + Z^{6} + Z^{5} + Z^{4} + Z) \]

- Can you tell of what degree (i.e. size of the kernel)?
- Can you compute the isogeny?
Plan

1. Transposition principle
2. Artin-Schreier towers
3. Isogenies
The transposition principle

“Let $\mathcal{P}$ be an arbitrary set. To any $R$-algebraic algorithm $A$ computing a family of linear functions $(f_p : M \to N)_{p \in \mathcal{P}}$ corresponds an $R$-algebraic algorithm $A^*$ computing the dual family $(f^*_p : N^* \to M^*)_{p \in \mathcal{P}}$. The algebraic time and space complexities of $A^*$ are bounded by the time complexity of $A$.”
The dual of a diagram

\[ A \rightarrow B \]
\[ C \rightarrow D \rightarrow E \]

\[ f \]
\[ g \]
\[ h \]

\[ A^* \leftarrow B^* \]
\[ C^* \leftarrow D^* \leftarrow E^* \]

\[ (f \circ g \circ h)^* = h^* \circ g^* \circ f^*; \]
\[ * \text{ is contravariant;} \]
\[ \text{A classical example is transposition of matrices: } (AB)^\top = B^\top A^\top; \]
\[ \text{From an algorithmic point of view, the number of arrows is a measure of complexity, and it is preserved under dualization.} \]
Transposition of arithmetic circuits

Arithmetic circuits are like diagrams enriched with a product. In particular they can be transposed:

\[ x_1 \oplus x_2 \oplus x_3 \leftrightarrow x_1 \ast 1 x_2 \ast 2 x_3 \]

This can be made precise using category theory.

\[ y_1 = x_1 + 3x_2 \]
\[ y_2 = x_3 \]

\[
\begin{pmatrix}
1 & 3 & 0 \\
0 & 0 & 1
\end{pmatrix}
\]
Transposition of arithmetic circuits

Arithmetic circuits are like diagrams enriched with a *product*. In particular they can be *transposed*:

\[
\begin{align*}
\chi_1 + \chi_2 &\rightarrow y_1 + y_2 \\
\chi_1 \& \chi_2 &\rightarrow y_1 \times 2
\end{align*}
\]

\[
\begin{align*}
\chi_1^* + \chi_2^* &\rightarrow y_1^* + y_2^* \\
\chi_1^* \& \chi_2^* &\rightarrow y_1^* \times 2
\end{align*}
\]

This can be made precise using category theory.

\[
y_1 = \chi_1 + 3\chi_2 \\
y_2 = \chi_3
\]

\[
\begin{pmatrix}
1 & 3 & 0 \\
0 & 0 & 1 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
1 & 0 \\
3 & 0 \\
0 & 1 \\
\end{pmatrix}
\]
Transposition of arithmetic circuits

Arithmetic circuits are like diagrams enriched with a product. In particular they can be transposed:

\[ x_1 \times x_2 \times x_3 + 2 \]

\[ y_1 = x_1 + 3x_2 \]

\[ y_2 = x_3 \]

\[
\begin{pmatrix}
1 & 3 & 0 \\
0 & 0 & 1 \\
3 & 0 & 0
\end{pmatrix}
\]

This can be made precise using category theory.
Transposition of straight line programs

Straight line programs = Arithmetic circuits

\begin{align*}
 a[1] &= a[0] + a[1] \\
 a[0] &= 0 \\
 a[1] &= 0 \\
 &\vdotswithin{=} \\
 a[n-1] &= a[n-2] + a[n-1] \\
 a[n-2] &= 0
\end{align*}

\begin{align*}
 a[0] &= a[0] + a[1] \\
 a[1] &= 0 \\
 a[0] &= a[0] + a[1] \\
 a[0] &= a[0] + a[1] \\
 &\vdotswithin{=} \\
 a[0] &= 0 \\
 a[0] &= a[0] + a[1]
\end{align*}

\[
\begin{pmatrix}
0 & \ldots & 0 \\
\vdots & \vdots & \vdots \\
0 & \ldots & 0 \\
1 & \ldots & 1
\end{pmatrix}
\quad
\begin{pmatrix}
0 & \ldots & 0 & 1 \\
\vdots & \vdots & \vdots & \vdots \\
0 & \ldots & 0 & 1
\end{pmatrix}

Programs = Families of straight line programs
**Automatic transposition?**

- Algorithms are hard to transpose, transposed algorithms are hard or impossible to understand;
- How to be confident that a transposed algorithm is well implemented if no one understands it?
- When proving programs with a proof assistant, why should we do the work twice?

**Previous work**

- Originally discovered in *electrical network theory* by Bordewijk 1957 (only works for $\mathbb{C}$); some authors attribute the discovery to Tellegen, Bordewijk’s director, but this is debated;
- Fiduccia 1973 and Hopcroft and Musinski 1973: transposition of *bilinear chains*, the most complete formulation (non-commutative rings);
- Special case of *automatic differentiation* Baur and Strassen 1983;
- In *computer algebra*, popularized by Shoup, von zur Gathen, Kaltofen, . . .
- Bostan, Lecerf, and Schost 2003 improve algorithms for polynomial evaluation and solve an open question on space complexity.
Does it make sense to transpose \( c := a \ast b \)?

- Most applications require to linearize a multi-linear program.
- Can we automatically deduce any possible linearisation of a program?
- Type inference systems can help us.
Suppose given a type $R$ implementing a ring. We want to define types $L$ (for *linear*) and $S$ (for *scalar*) such that the following equations hold:

\[
\begin{align*}
\text{plus} &: \ L \to L \to L \\
\text{plus} &: \ S \to S \to S \\
\text{times} &: \ L \to S \to L \\
\text{times} &: \ S \to L \to L \\
\text{times} &: \ S \to S \to S \\
\text{zero} &: \ L \\
\text{zero} &: \ S \\
\text{one} &: \ S 
\end{align*}
\]
Suppose given a type R implementing a ring. We want to define types L (for *linear*) and S (for *scalar*) such that the following equations hold

\[
\begin{align*}
\text{plus} & : L \to L \to L \\
\forall \alpha \in \{L, S\}. \alpha \to \alpha \to \alpha \\
\text{plus} & : S \to S \to S \\
\text{times} & : L \to S \to L \\
\forall \alpha \in \{L, S\}. \alpha \to S \to \alpha \\
\text{times} & : S \to L \to L \\
\text{times} & : S \to S \to S \\
\text{zero} & : L \\
\forall \alpha \in \{L, S\}. \alpha \\
\text{zero} & : S \\
\text{one} & : S 
\end{align*}
\]
The solution in Haskell

```haskell
data L = L R
data S = S R

class Ring r where
    zero :: r
    (<+>) :: r -> r -> r
    neg :: r -> r
    (<*>) :: r -> S -> r

    one = S oneR
    (S a) == (S b) = a == b
```

To treat `times :: S -> L -> L`, we extend the Hindley-Milner type inference
to handle lists of acceptable unifications.
We are implementing

A Python-like ad-hoc language, compiled/interpreted in Python, featuring:

- Algebraic constructs (Rings, Modules, Fields, ...);
- Transposition of multilinear/recursive code;
- Parameterizable linearity inference (including commutative multiplication);
- Algebraic complexity preserving;
- Easily used on top of Computer Algebra Systems that have a Python interface;
- Other Computer Algebra Systems will be able to work with it as we will add more languages to the output of the compiler (OCaml and Haskell look easy, C is somewhat harder).

http://transalpyne.gforge.inria.fr/

---

Coding

Integration of automatic transposition in a Computer Algebra System. (Sage? Mathemagix?)

Arithmetic circuits and categorical semantics

Joint work with M. Boespflug:
- We have implemented a Domain Specific Language in Haskell,
- the result is not satisfactory due to Haskell’s lack of support for dependent types.

Automated Theorem Provers

We plan to write a library to ease the use of the transposition principle in Automated Theorem Provers. (Coq? Agda? Isabelle?)
Plan

1. Transposition principle

2. Artin-Schreier towers

3. Isogenies
Newton sums

Newton identities

- Given a polynomial \( f = \prod_j (X - \alpha_j) \in \mathbb{K}[X] \),
- The Newton sums are the \( p_i = \sum_j \alpha_j^i \) for any \( i \geq 0 \)

\[
\frac{f'}{f} = \sum_{i \geq 0} \frac{p_i}{T^{i+1}} \quad \Leftrightarrow \quad f = \exp \left( \int \frac{f'}{f} \right) = T^d \exp \left( - \sum_{i \geq 1} \frac{p_i}{iT^i} \right).
\]
Newton sums

Newton identities

- Given a polynomial \( f = \prod_j (X - \alpha_j) \in \mathbb{K}[X] \),
- The Newton sums are the \( p_i = \sum_j \alpha_j^i \) for any \( i \geq 0 \)

\[
\frac{f'}{f} = \sum_{i \geq 0} \frac{p_i}{T_i^{i+1}} \quad \iff \quad f = \exp \left( \int \frac{f'}{f} \right) = T^d \exp \left( - \sum_{i \geq 1} \frac{p_i}{iT_i} \right).
\]

Trace formulas

Let \( \mathcal{A} = \mathbb{K}[X]/f(X) \), then

\[
p_i = \text{Tr}_{\mathcal{A}/\mathbb{K}} X^i.
\]

More generally for any \( a, z \in \mathcal{A} \), with \( z \) primitive and \( g \) its minimal polynomial

\[
\sum_{i \geq 0} \frac{a \cdot \text{Tr}_{\mathcal{A}/\mathbb{K}} z^i}{T_i^{i+1}} = \sum_{i \geq 0} \frac{\text{Tr}_{\mathcal{A}/\mathbb{K}} az^i}{T_i^{i+1}} = \frac{A(T)}{g(T)} \quad \text{and} \quad a = \frac{A(z)}{g'(z)}.
\]
Shoup’s algorithm (Shoup 1995, 1999)

**Polynomial evaluation:** $k[T] \rightarrow K/k$

$$g \mapsto g(\sigma)$$

**Power projection:** $(K/k)^* \rightarrow k[T]^*$

$$\ell \mapsto \sum_{i>0} \frac{\ell(\sigma^i)}{T^i}$$

**Power projection = transposed polynomial evaluation**

Let $A = K[X]/f(X)$ and $z \in A$. Take any algorithm that computes $g \mapsto g(z)$ and transpose it:

- Apply to $\text{Tr}_{A/K}$ to compute the characteristic polynomial of $z$;
- Apply to $\alpha \cdot \text{Tr}_{A/K}$ to compute a representation of $\alpha$ as a univariate polynomial in $z$.

The complexity of the original algorithm is preserved by the transposition principle!
Rational Univariate Representation

Generalization in many variables (Giusti, Lecerf, and Salvy 2001; Rouillier 1999)

Let $\mathcal{A} = \mathbb{K}[x_1, \ldots, x_n]/I$ and $z \in \mathcal{A}$

\[ g(z) = 0, \]
\[ x_1 = \frac{g_1(z)}{g(z)}, \]
\[ \vdots \]
\[ x_n = \frac{g_n(z)}{g(z)}, \]

Change of basis

These two operations have the same cost, by the transposition principle:

- Going from the univariate basis
  \( Z = \{1, z, \ldots, z^{d-1}\} \)

  to any basis $\mathcal{B}$ is equivalent to polynomial evaluation in $z$.

- Going from $\mathcal{B}$ to $Z$ is equivalent to Rational Univariate Representation.
Application to towers of extension fields

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

\[ U_1 = \frac{U_0[X_1]}{P_0(X_1)} \]

\[ U_0 = \mathbb{F}_p[\mathbb{F}_p[X_0]] = \frac{\mathbb{F}_p[X_0]}{Q(X_0)} \]

Change of basis

\[ Z = \{1, X_k, X_k^2, \ldots\} \]
\[ B = \{1, X_{k-1}, X_{k-1}, \ldots, X_k, X_{k-1}X_k, X_{k-1}^2X_k, \ldots\} \]

\[ \begin{align*}
Q_k(X_k) &= 0 \\
X_{k-1} &= \frac{R(X_k)}{Q_k'(X_k)}
\end{align*} \leftrightarrow \begin{align*}
P_{k-1}(X_k, X_{k-1}) &= 0 \\
Q_{k-1}(X_{k-1}) &= 0
\end{align*} \]

- Multiplication is faster on \( Z \);
- Embeddings are faster on \( B \);
- A fast algorithm for \( Z \to B \) implies a fast one for \( B \to Z \).
Application to Artin-Schreir towers

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

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

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

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

\[ \mathbb{L}/\mathbb{K} \text{ of characteristic } p \text{ such that} \]

\[ \mathbb{L} = \mathbb{K}[X]/(X^p - X - \alpha). \]

Artin-Schreier extension

Our construction

Let \( x_0 = X_0 \) such that \( \text{Tr}_{\mathbb{U}_0/\mathbb{F}_p}(x_0) \neq 0 \), let

\[ P_0 = X^p - X - x_0 \]

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

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

This tower is such that \( x_i \) generates \( \mathbb{U}_i/\mathbb{F}_p \).

Application to Artin-Schreir towers

The algorithms

All of these operations can be done in quasi-optimal time and space (w.r.t. the size of $U_k$):

- Minimal polynomials of $x_i$ over $\mathbb{F}_p$ computed iteratively;
- Change $Z \rightarrow B$ using a $p$-ary divide-and-conquer;
- Change $B \rightarrow Z$ by trace formulas + transposed algorithms;
- Fast univariate multiplication via FFT, fast arithmetics (inversion, GCD, ...);
- Traces and pseudotraces, Frobenius morphisms;

Implementation

- C++ with NTL implementation released under GPL: http://www.lix.polytechnique.fr/~defeo/FAAST/
- Port to SAGE one day?
1. Transposition principle

2. Artin-Schreier towers

3. Isogenies
Isogenies between elliptic curves

\[ J : E \to E' \]

(Separable) isogeny: (separable) non-constant rational morphism preserving the point at infinity.

Properties

- Finite kernel, surjective (in \( \bar{\mathbb{K}} \));
- Defined by rational fractions with a pole at infinity;
- \( \#E(\mathbb{F}_{q^n}) = \#E'(\mathbb{F}_{q^n}) \) for every \( n \),
- Dual isogeny: \([m] = J \circ \hat{J}\).

Multiplication

\[ [m] : E(\bar{\mathbb{K}}) \to E(\bar{\mathbb{K}}) \]

\[ P \mapsto [m]P \]

\( \ker J = E[m], \quad \deg J = m^2. \)
Isogenies between elliptic curves

\[ J : E \rightarrow E' \]

(Separable) isogeny: (separable) non-constant rational morphism preserving the point at infinity.

**Properties**
- Finite kernel, surjective (in \( \bar{\mathbb{K}} \));
- Defined by rational fractions with a pole at infinity;
- \( \#E(\mathbb{F}_{q^n}) = \#E'(\mathbb{F}_{q^n}) \) for every \( n \),
- Dual isogeny: \([m] = J \circ \hat{J}\).

**Frobenius endomorphism**
\[ \varphi : E(\bar{\mathbb{K}}) \rightarrow E(\bar{\mathbb{K}}) \]
\[ (X, Y) \mapsto (X^q, Y^q) \]
\[ \ker \varphi = \{0\}, \quad \deg J = q. \]
Isogenies between elliptic curves

\[ \mathcal{J} : E \to E' \]

**Separable isogeny**: (separable) non-constant rational morphism preserving the point at infinity.

**Properties**

- Finite kernel, surjective (in \( \bar{\mathbb{K}} \));
- Defined by rational fractions with a pole at infinity;
- \( \#E(\mathbb{F}_{q^n}) = \#E'(\mathbb{F}_{q^n}) \) for every \( n \),
- Dual isogeny: \([m] = \mathcal{J} \circ \hat{\mathcal{J}}\).

**Separable isogeny, odd degree (simplified Weierstrass model)**

\[ \mathcal{J}(X, Y) = \left( \frac{g(X)}{h^2(X)}, cY \left( \frac{g(X)}{h^2(X)} \right)' \right) \]

\( \ell = \deg \mathcal{J} = \# \ker \mathcal{J} = 2 \deg h + 1 \) odd.
Vélu formulas

Vélu 1971 (algebraically closed field)

Given the kernel $H$, computes $J : E \rightarrow E/H$ given by

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

$$J(P) = \left( x(P) + \sum_{Q \in H^*} x(P + Q) - x(Q), y(P) + \sum_{Q \in H^*} y(P + Q) - y(Q) \right).$$

For $p \geq 3$, given $h(x)$ vanishing on $H$

$$y^2 = f(x), \quad t = \sum_{Q \in H^*} f'(Q), \quad u = \sum_{Q \in H^*} 2f(Q), \quad w = u + \sum_{Q \in H^*} x(Q)f'(Q),$$

$$J(x, y) = \left( \frac{g(x)}{h(x)}, y \left( \frac{g(x)}{h(x)} \right)' \right) \quad \text{avec} \quad \frac{g(x)}{h(x)} = x + t \frac{h'(x)}{h(x)} - u \left( \frac{h'(x)}{h(x)} \right)'$$
Isogeny computation

Given $E, E', \ell$, compute $J : E \to E'$

By Vélu formulas: $J(x, y) = \left( \frac{g(x)}{h(x)}, cy \left( \frac{g(x)}{h(x)} \right)' \right)$, hence

$$c^2(x^3 + ax + b) \left( \frac{g(x)}{h(x)} \right)'^2 = \left( \frac{g(x)}{h(x)} \right)^3 + a' \frac{g(x)}{h(x)} + b'$$

BMSS algorithm (Bostan, Morain, Salvy, and Schost 2008)

1. Change variables $S(x) = \sqrt{\frac{h(1/x^2)}{g(1/x^2)}} \iff \frac{g(x)}{h(x)} = \frac{1}{S(1/\sqrt{x})^2}$;
2. Power series solution of $c^2(bx^6 + ax^4 + 1)S'/2 = 1 + a'S^4 + b'S^6$;
3. Inverse the change of variables, reconstruct a rational fraction.

Lercier and Sirvent 2008

When $p$ exceeds the precision, a division by zero happens:

- Lift $E$ and $E'$ in the $p$-adics while keeping $\Phi_\ell \left( j(\tilde{E}), j(\tilde{E}') \right) = 0$;
- Apply BMSS in $\mathbb{Q}_q$. 
Couveignes’ algorithms

Idea: Send $E[p^k]$ over $E'[p^k]$

Couveignes 1994

- Compute the extensions $U_i/F_q$ such that $E[p^i]$ is defined in $U_i$;
- Pick $k$ large enough ($k \sim \log_p 4\ell$);
- Compute $P$, a generator of $E[p^k]$;
- Compute $P'$, a generator of $E'[p^k]$;
- Compute the polynomial $T$ vanishing $E[p^k]$;
- Interpolate $A: \chi(P) \mapsto \chi(P')$;
- Reconstruct a rational fraction $\frac{g}{h} \equiv A \mod T$;
- If $\frac{g}{h}$ is an isogeny, done; otherwise pick another $P'$.

Couveignes 1996

- Compute the extensions $U_i/F_q$ such that $E[p^i]$ is defined in $U_i$;
- Pick $k$ large enough ($k \sim \log_p 4\ell$);
- Compute $P$, a generator of $E[p^k]$;
- Compute $P'$, a generator of $E'[p^k]$;
- Compute the polynomial $T$ vanishing $E[p^k]$;
- Interpolate $A: \chi(P) \mapsto \chi(P')$;
- Reconstruct a rational fraction $\frac{g}{h} \equiv A \mod T$;
- If $\frac{g}{h}$ is an isogeny, done; otherwise pick another $P'$.
Couveignes’ algorithms

**Idea:** Send $E[p^k]$ over $E'[p^k]$

### Couveignes 1994
- Work in the formal group $E$ of $E$: a *formal point* is a series in a formal parameter $\tau$;
- Fix a precision *large enough* for $\mathbb{F}_q[[\tau]]$ ($\sim \log_p 4\ell$);
- Compute a morphism $U(\tau) : E \to E'$;
- Reconstruct a rational fraction \( \frac{g(X)}{h(X)} = \frac{1}{U(1/X)} \);
- If $\frac{g}{h}$ is an isogeny, done; otherwise pick another $U$.

### Couveignes 1996
- Compute the extensions $U_i/\mathbb{F}_q$ such that $E[p^i]$ is defined in $U_i$;
- Pick $k$ *large enough* ($k \sim \log_p 4\ell$);
- Compute $P$, a generator of $E[p^k]$;
- Compute $P'$, a generator of $E'[p^k]$;
- Compute the polynomial $T$ vanishing $E[p^k]$;
- Interpolate $A : x(P) \mapsto x(P')$;
- Reconstruct a rational fraction $\frac{g}{h} \equiv A \mod T$;
- If $\frac{g}{h}$ is an isogeny, done; otherwise pick another $P'$. 
**Idea:** Send $E[p^k]$ over $E'[p^k]$

**Couveignes 1994**
- Work in the formal group $\mathcal{E}$ of $E$: a **formal point** is a series in a formal parameter $\tau$;
- Fix a precision *large enough* for $\mathbb{F}_q[[\tau]]$ ($\sim \log_p 4\ell$);
- Compute a morphism $U(\tau) : \mathcal{E} \rightarrow \mathcal{E}'$;
- Reconstruct a rational fraction $\frac{g(X)}{h(X)} = \frac{1}{U(1/X)}$;
- If $\frac{g}{h}$ is an isogeny, done; otherwise pick another $U$.
- $U$ is uniquely determined by its action on $\mathcal{E}[p^k]$ for every $k$.

**Couveignes 1996**
- Compute the extensions $U_i/\mathbb{F}_q$ such that $E[p^i]$ is defined in $U_i$;
- Pick $k$ *large enough* ($k \sim \log_p 4\ell$);
- Compute $P$, a generator of $E[p^k]$;
- Compute $P'$, a generator of $E'[p^k]$;
- Compute the polynomial $T$ vanishing $E[p^k]$;
- Interpolate $A : x(P) \mapsto x(P')$;
- Reconstruct a rational fraction $\frac{g}{h} \equiv A \mod T$;
- If $\frac{g}{h}$ is an isogeny, done; otherwise pick another $P'$. 
Fast Couveignes 1996

- Compute the extensions $\mathbb{U}_i/\mathbb{F}_q$ such that $E[p^i]$ is defined in $\mathbb{U}_i$;
- Pick $k$ large enough $(k \sim 4\ell)$;
- Compute $P$, a generator of $E[p^k]$;
- Compute $P'$, a generator of $E'[p^k]$;
- Compute the polynomial $T$ vanishing $E[p^k]$;
- Interpolate $A : x(P) \mapsto x(P')$;
- Reconstruct a rational fraction $\frac{g}{h} \equiv A \mod T$;
- If $\frac{g}{h}$ is an isogeny, done; otherwise pick another $P'$.

---

• Compute the extensions $U_i/F_q$ such that $E[p^i]$ is defined in $U_i$;
• Pick $k$ large enough ($k \sim 4\ell$);
• Compute $P$, a generator of $E[p^k]$;
• Compute $P'$, a generator of $E'[p^k]$;
• Compute the polynomial $T$ vanishing $E[p^k]$;
• Interpolate $A : x(P) \mapsto x(P')$;
• Reconstruct a rational fraction $\frac{g}{h} \equiv A \mod T$;
• If $\frac{g}{h}$ is an isogeny, done; otherwise pick another $P'$.

---

Fast Colveignes 1996$^3$

- Compute the extensions $\mathbb{U}_i/\mathbb{F}_q$ such that $E[p^i]$ is defined in $\mathbb{U}_i$;
- Pick $k$ large enough ($k \sim 4\ell$);
- Compute $P$, a generator of $E[p^k]$;
- Compute $P'$, a generator of $E'[p^k]$;
- Compute the polynomial $T$ vanishing $E[p^k]$;
- Interpolate $A : x(P) \mapsto x(P')$;
- Reconstruct a rational fraction $\frac{g}{h}$ $\equiv A \mod T$;
- If $\frac{g}{h}$ is an isogeny, done; otherwise pick another $P'$.

An Artin-Schreir tower: $\tilde{O}(\ell)$

An isomorphism of Artin-Schreier towers: $\tilde{O}(\ell)$
An isomorphism of Artin-Schreier towers: $\tilde{O}(\ell)$

---

Fast Couveignes 1996\textsuperscript{3}

- Compute the extensions $\mathbb{U}_i/\mathbb{F}_q$ such that $E[p^i]$ is defined in $\mathbb{U}_i$;
- Pick $k$ \textit{large enough} ($k \sim 4\ell$);
- Compute $P$, a generator of $E[p^k]$;
- Compute $P'$, a generator of $E'[p^k]$;
- Compute the polynomial $T$ vanishing $E[p^k]$;
- Interpolate $A : x(P) \mapsto x(P')$;
- Reconstruct a rational fraction $\frac{g}{h} \equiv A \mod T$;
- If $\frac{g}{h}$ is an isogeny, done; otherwise pick another $P'$.

An Artin-Schreir tower: $\tilde{O}(\ell)$

An isomorphism of Artin-Schreier towers: $\tilde{O}(\ell)$

An isomorphism of Artin-Schreier towers: $\tilde{O}(\ell)$

Fast interpolation in towers of extensions: $\tilde{O}(\ell)$

---

Fast Couveignes 1996

- Compute the extensions $U_i/F_q$ such that $E[p^i]$ is defined in $U_i$;
- Pick $k$ large enough ($k \sim 4\ell$);
- Compute $P$, a generator of $E[p^k]$;
- Compute $P'$, a generator of $E'[p^k]$;
- Compute the polynomial $T$ vanishing $E[p^k]$;
- Interpolate $A : x(P) \mapsto x(P')$;
- Reconstruct a rational fraction $\frac{g}{h} \equiv A \mod T$;
- If $\frac{g}{h}$ is an isogeny, done; otherwise pick another $P'$.

An Artin-Schreir tower: $\tilde{O}(\ell)$

An isomorphism of Artin-Schreier towers: $\tilde{O}(\ell)$

An isomorphism of Artin-Schreier towers: $\tilde{O}(\ell)$

Fast interpolation in towers of extensions: $\tilde{O}(\ell)$

XGCD: $\tilde{O}(\ell)$

---

• Compute the extensions \( U_i/F_q \) such that \( E[p^i] \) is defined in \( U_i \);
• Pick \( k \) large enough (\( k \sim 4\ell \));
• Compute \( P \), a generator of \( E[p^k] \);
• Compute \( P' \), a generator of \( E'[p^k] \);
• Compute the polynomial \( T \) vanishing \( E[p^k] \);
• Interpolate \( A : x(P) \mapsto x(P') \);
• Reconstruct a rational fraction \( \frac{g}{h} \equiv A \mod T \);
• If \( \frac{g}{h} \) is an isogeny, done; otherwise pick another \( P' \).

An Artin-Schreir tower: \( \tilde{O}(\ell) \)

An isomorphism of Artin-Schreier towers: \( \tilde{O}(\ell) \)
An isomorphism of Artin-Schreier towers: \( \tilde{O}(\ell) \)

Fast interpolation in towers of extensions: \( \tilde{O}(\ell) \)

XGCD: \( \tilde{O}(\ell) \)

Repeat \( O(\ell) \) times

---

How to recognize an isogeny?

- **Degree:** \( \frac{g}{h} \) with \( \deg g = \ell, \deg h = \ell - 1 \); \( O(1) \)
- **Square factor:** \( h = \prod_{Q \in H^*} (X - x(Q)) = f^2 \) if \( \ell \) odd; \( \tilde{O}(\ell) \)
- **Group action:** Test with random points; \( O(\ell) \)
- **Factor of the \( \ell \)-division polynomial:** Compute \( \phi_\ell \mod h \). \( \tilde{O}(\ell) \)
How to recognize an isogeny?

\[ AU_i + TV_i = R_i \iff A \equiv \frac{R_i}{U_i} \mod T \]

\[ \ell = 11 \]
How to recognize an isogeny?

\[ A U_i + T V_i = R_i \iff A \equiv \frac{R_i}{U_i} \mod T \]

\[ \ell = 11 \]

<table>
<thead>
<tr>
<th>( \deg R_i )</th>
<th>( \deg U_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>3141592653589793238462643</td>
<td>0</td>
</tr>
</tbody>
</table>
How to recognize an isogeny?

\[ AU_i + TV_i = R_i \iff A \equiv \frac{R_i}{U_i} \mod T \]

\[ \ell = 11 \]

\[
\begin{array}{l|l}
\text{deg } R_i & \text{deg } U_i \\
3141592653589793238462643 & 0 \\
3141592653589793238462642 & 1 \\
\end{array}
\]
How to recognize an isogeny?

\[ AU_i + TV_i = R_i \iff A \equiv \frac{R_i}{U_i} \mod T \]

\[ \ell = 11 \]

<table>
<thead>
<tr>
<th>( \deg R_i )</th>
<th>\hline</th>
<th>( \deg U_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>3141592653589793238462643</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3141592653589793238462642</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3141592653589793238462641</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
How to recognize an isogeny?

\[ AU_i + TV_i = R_i \quad \Leftrightarrow \quad A \equiv \frac{R_i}{U_i} \mod T \]

\[ \ell = 11 \]

\[
\begin{array}{l|l}
\text{deg } R_i & \text{deg } U_i \\
3141592653589793238462643 & 0 \\
3141592653589793238462642 & 1 \\
3141592653589793238462641 & 2 \\
\vdots & \vdots \\
3141592653589793238462634 & 9 \\
\end{array}
\]
How to recognize an isogeny?

\[ \Lambda U_i + TV_i = R_i \iff \Lambda \equiv \frac{R_i}{U_i} \mod T \]

\[ \ell = 11 \]

<table>
<thead>
<tr>
<th>( \deg R_i )</th>
<th>( \deg U_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>3141592653589793238462643</td>
<td>0</td>
</tr>
<tr>
<td>3141592653589793238462642</td>
<td>1</td>
</tr>
<tr>
<td>3141592653589793238462641</td>
<td>2</td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
</tr>
<tr>
<td>3141592653589793238462634</td>
<td>9</td>
</tr>
</tbody>
</table>
How to recognize an isogeny?

\[ A U_i + TV_i = R_i \quad \iff \quad A \equiv \frac{R_i}{U_i} \mod T \]

\[ \ell = 11 \]

\[
\begin{array}{c|c}
\deg R_i & \deg U_i \\
3141592653589793238462643 & 0 \\
3141592653589793238462642 & 1 \\
3141592653589793238462641 & 2 \\
\vdots & \vdots \\
3141592653589793238462634 & 9 \\
11 & 10
\end{array}
\]
How to recognize an isogeny?

\[ AU_i + TV_i = R_i \iff A \equiv \frac{R_i}{U_i} \mod T \]

\[ \ell = 11 \]

<table>
<thead>
<tr>
<th>( \text{deg } R_i )</th>
<th>( \text{deg } U_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>3141592653589793238462643</td>
<td>0</td>
</tr>
<tr>
<td>3141592653589793238462642</td>
<td>1</td>
</tr>
<tr>
<td>3141592653589793238462641</td>
<td>2</td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
</tr>
<tr>
<td>3141592653589793238462634</td>
<td>9</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>3141592653589793238462633</td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
</tr>
</tbody>
</table>
Isogenies of unknown degree

- This pattern is extremely rare.
- This is the only phase of Couveignes’ algorithms that depends on $\ell$. 

Actually, this does not really depend on $\ell$, just on the existence of a gap. If $\ell$ is not known in advance, it is enough to look for a gap. Thus, any isogeny of degree $\ll p$ can be obtained with one single run of Couveignes’ algorithms.
Isogenies of unknown degree

- This pattern is extremely rare.
- This is the only phase of Couveignes’ algorithms that depends on $\ell$.
- Actually, this does not really depend on $\ell$, just on the existence of a gap.
- If $\ell$ is not known in advance, it is enough to look for a gap.
- Thus, any isogeny of degree $\ll p^k$ can be obtained with one single run of Couveignes’ algorithms.
Perspectives

Looking for the quasi-linear complexity

- The Weierstrass model has a canonicity defect: use other parameterizations? Formal groups?
- How to obtain local information on the behavior of the isogeny? (for example, its action on $E[p]$)

Isogenies of unknown degree

- This variant of Couveignes 1996 is at the moment the fastest (both in theory and in practice) algorithm for this task.
- We tested two curves over $\mathbb{F}_{2^{161}}$, isogenous of unknown degree, taken from Teske 2006;
- Certified in 258 cpu-hours that no isogeny of degree $2^c \ell$ for any $c$ and $\ell < 2^{11}$ exists;
- Certified in 1195 cpu-hours that no isogeny of degree less than $2^{12}$ exists.
- The two curves have an isogeny of (very smooth) degree $\sim 2^{1050}$. Proving that no isogeny of smaller degree exists is momentarily out of reach.
Fast Algorithms for Towers of Finite Fields and Isogenies

13 décembre, École Polytechnique
heure et amphi à préciser


References VI

“Tellegen’s principle into practice.”
In: ISSAC ’03: Proceedings of the 2003 international symposium on Symbolic and algebraic computation.
Philadelphia, PA, USA: ACM,
Pp. 37–44.
URL: http://dx.doi.org/10.1145/860854.860870.

De Feo, Luca and Éric Schost (2010).
“transalpyne: a language for automatic transposition.”
In: SIGSAM Bulletin 44.1/2,
URL: http://dx.doi.org/10.1145/1838599.1838624.

“A new polynomial factorization algorithm and its implementation.”
In: Journal of Symbolic Computation 20.4,


