Point counting in genus 2: reaching 128 bits

P. Gaudry — **Cacao project** — **CNRS-INRIA**

É. Schost — **ORCCA** — **UWO**

Thanks to Dan Bernstein and Nikki Pitcher

Genus 2 curves and associated objects

In what follows:

- C is the curve defined over $\mathbb{F}_p$ by

$$y^2 = x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0,$$

  with p large prime.

- $\mathbf{J}$ is its Jacobian
  - variety of dimension 2;
  - we will work in Mumford coordinates.

- $\mathbf{K}$ is the associated Kummer surface
  - $\mathbf{K} = \mathbf{J}$ after identifying opposite points;
  - a variety of dimension 2 too;
  - we won't work with it too much.

Our question

Finding a curve

- whose Jacobian and its twist have an almost prime cardinality;
- over a prime field;
- with small coefficients;
  - the coefficients defining the Kummer surface should be small integers, to make scalar multiplication fast.
- with $p = 2^{127} - 1$.

We are not there yet, but almost.

- A first 128 bit run.
- The curve was rather random, but slightly favorable.

Previous work, large characteristic

Schoof (1985): polynomial time algorithm for elliptic curves.

- Pila (1990): algorithm for abelian varieties.
- Kampkötter (1991): genus 2 algorithm.
- Adleman-Huang (1996), Huang-Ierardi (1998): improvements of Pila's work.
- Gaudry-Harley (2000): genus 2 algorithm, $p \simeq 2^{61}$.
- Gaudry-S. (2004): cryptographic size: $p \simeq 2^{82}$.

Baby steps / giant steps

- Matsuo-Chao-Tsujii (2002): efficient strategy.
- Gaudry-S. (2004): parallel, low-memory version of Matsuo-Chao-Tsujii.

Sutherland (2007)

- curves whose twist has a smooth order.

Schoof's approach

Let

$$\chi = T^4 - s_1 T^3 + s_2 T^2 - p s_1 T + p^2 \in \mathbb{Z}[T]$$

be the characteristic polynomial of the Frobenius endomorphism on $\mathbf{J}$.

- $\mathrm{card}(\mathbf{J}) = \chi(1)$;
- for $\ell \in \mathbb{N}$, computing the ℓ-torsion (or a subset of it) gives $\chi \bmod \ell$ (up to some indeterminacy, maybe).

General scheme:

- for as many coprimes $\ell_1, \ldots, \ell_r$ as possible, compute the ℓ-torsion;
- some collision detection technique is used if we do not have enough precision to conclude by Chinese remaindering:

  If $\ell_1 \cdots \ell_r = m$, then the cost is about $p^{0.75}/m$.

Concretely

It boils down to solving polynomial systems.

Some numbers:

- an element of the Jacobian has **4** coordinates with **2** relations.
- ℓ-torsion has cardinality ℓ^4.

Large primes: up to $\ell = 31$ or $\ell = 37$ ($\ell = 43$ doable?)

- bivariate resultants.

Prime powers:

- nice improvements on 2^k-torsion and 3^k-torsion;
- dull improvements on 5^k-torsion and 7^k-torsion.

Concretely

Software environment: NTL

- does better than Magma for the routines we need
  - most basic routines on uni (bi, tri) -variate polynomials.
- convenient
- on the other hand, no Gröbner engine
  - anyway, faster workarounds.

Large primes

Reduction to bivariate solving

Mostly from Gaudry-Harley and Gaudry-S.:

- Rewrite $[\ell]D = 0$ as

$$D = P_1(x_1, y_1) + P_2(x_2, y_2), \qquad [\ell]P_1 = -[\ell]P_2.$$

- You get equations in (x_1, y_1, x_2, y_2) with symmetries.
- Rewrite these equations in the elementary symmetric polynomials.
  Saves a factor of 2.
- Bivariate equations: bivariate resultants.
- Output size $\simeq \ell^4$, cost $O^{\tilde{}}(\ell^6)$ operations in $\mathbb{F}_p$.
  $O^{\tilde{}}$ means we neglect logarithmic factors.

What's left to improve:

- Bivariate resultants are sub-optimal.
- Systems are over-determined, but we don't know how to exploit it.

Lifting the 2-torsion

Lifting the torsion

`While (possible==true) do`

- write the equations that say $[\ell]P_{k+1} = P_k$ ℓ^4 solutions;
- extend the base field with one solution;
- continue. $\ell \to \ell^2 \to \ell^3 \to \cdots$

Here, we deal with $\ell = 2, 3, 5, 7$

- general techniques (Gröbner bases, resultants) do not perform very well;
- the systems are simple enough that specialized solutions may pay off:
  - $\ell = 2$: reduction to square-root extraction;
  - $\ell = 3$: deformation techniques & root-finding;
  - $\ell = 5, 7$: bivariate resultants, again.

Using the Kummer surface

Chudnovsky[2], Gaudry:

- fast formulas for scalar multiplication in $\mathbf{K}$;
- in particular, doubling: the coordinates of $[2](x, y, z, t)$ are obtained through a few linear combinations and squarings.

Consequence:

- division by 2 is done in $\mathbf{K}$ by taking 4 square roots; $2^4 = 16$
- the points in $\mathbf{K}$ are mapped back to $\mathbf{J}$.

Handling quadratic extensions

Fact

- Each division-by-2 doubles the degree of the current base field over $\mathbb{F}_p$ (after k steps, we are in a degree 2^k extension)

Possible data representations

Triangular	Primitive element
$\begin{cases} T_1(X_1) \\ \vdots \\ T_k(X_1, \ldots, X_k) \end{cases}$	$P(T) = 0, \quad \begin{cases} X_1 = V_1(T) \\ \vdots \\ X_k = V_k(T) \end{cases}$
$\deg(T_k, X_k) = 2$	$\deg(P, T) = 2^k$

Computations

1. We use a primitive element representation

- multiplications, inverses cost $O^{\sim}(2^k)$

2. Taking square roots requires some work:

- when no root exists, extend the base field.
- main subroutine: modular composition $A, B, C \mapsto A(B) \bmod C$.
- most other operations reduce to composition or a dual form of it.
  - irreducibility tests
  - finding new primitive elements
- cost: $O^{\sim}(2^{1.5k})$ (polynomial operations) $+ O(2^{2k})$ (linear algebra)

In detail

We start step k with

$$P(T) = 0, \quad \begin{cases} X_1 = V_1(T) \\ \quad \vdots \\ X_k = V_k(T) \end{cases} \qquad \deg(P, T) = 2^k,$$

and P irreducible. We want to find a square root of $A(T)$.

Facts: in real life,

- factoring in $\mathbb{F}_p[X]$ is fast;
- taking square roots in $\mathbb{F}_p[X]/P(X)$ is slow.

Our approach

1. Change the order.

$$\begin{cases} Y^2 - A(X) \\ P(X) \end{cases} \longrightarrow \begin{cases} X - B(Y) \\ Q(Y) \end{cases} \qquad \deg(Q) = 2\deg(P).$$

Nice case: Y is a primitive element.

Cost: similar to that of modular composition.

2. Factor.

- either Q is irreducible,
- or it has two factors of the same degree.

Cost: similar to that of modular composition, up to some log's.

3. Update.

Cost: similar to that of modular composition.

Lifting the 3-torsion

Tools required

For the 3-torsion, we found no nice formula as for $\ell = 2$.

Possible workarounds:

- Gröbner
- resultants
- something else

Remark:

- All solutions should have a cost of about $\tilde{O}(C(3^k))$, with $C(3^k)$ the cost of modular composition in degree 3^k.
- It's all in the constant.
- Upcoming: deformation techniques (Pardo-San Martin).

Deformation techniques

Basic idea

- The system $[3]P = Q$ is parametrized by the coordinates of Q.
- Set up a homotopy between the target $[3]P = Q$ and an initial system $[3]P_0 = Q_0$ for which we know the solutions

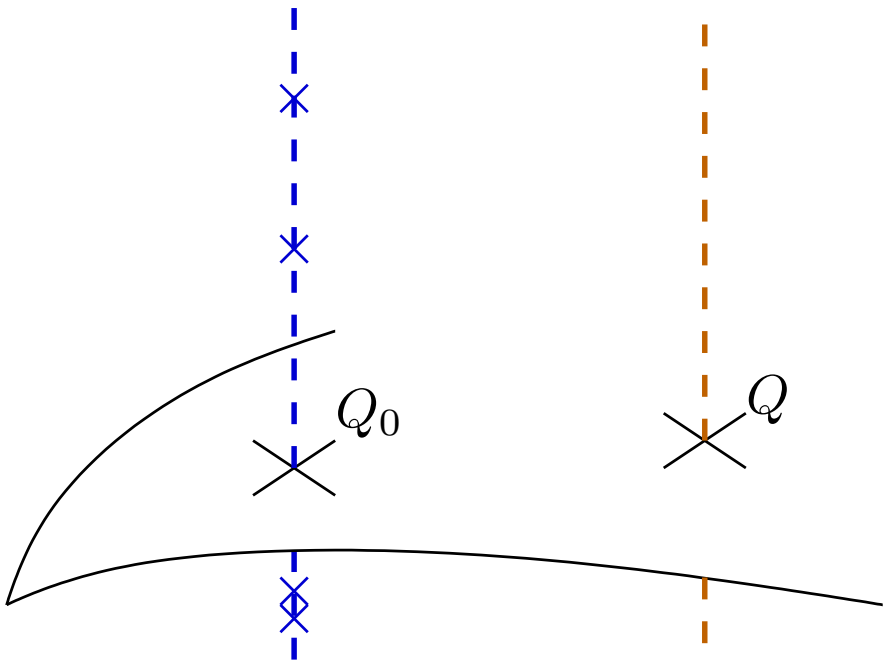
  basically, we let $Q_t = (1-t)Q_0 + tQ$.
- Compute a description of the solution curve and let $t = 1$.

Deformation techniques

Basic idea

- The system $[3]P = Q$ is parametrized by the coordinates of Q.
- Set up a homotopy between the target $[3]P = Q$ and an initial system $[3]P_0 = Q_0$ for which we know the solutions

  basically, we let $Q_t = (1-t)Q_0 + tQ$.
- Compute a description of the solution curve and let $t = 1$.

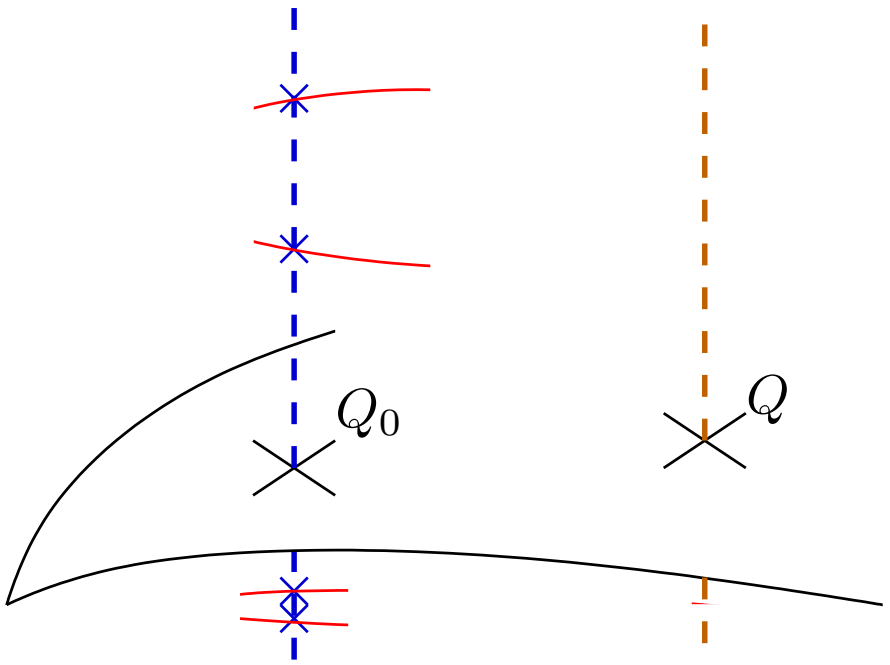


Deformation techniques

Basic idea

- The system $[3]P = Q$ is parametrized by the coordinates of Q.
- Set up a homotopy between the target $[3]P = Q$ and an initial system $[3]P_0 = Q_0$ for which we know the solutions

  basically, we let $Q_t = (1-t)Q_0 + tQ$.
- Compute a description of the solution curve and let $t = 1$.

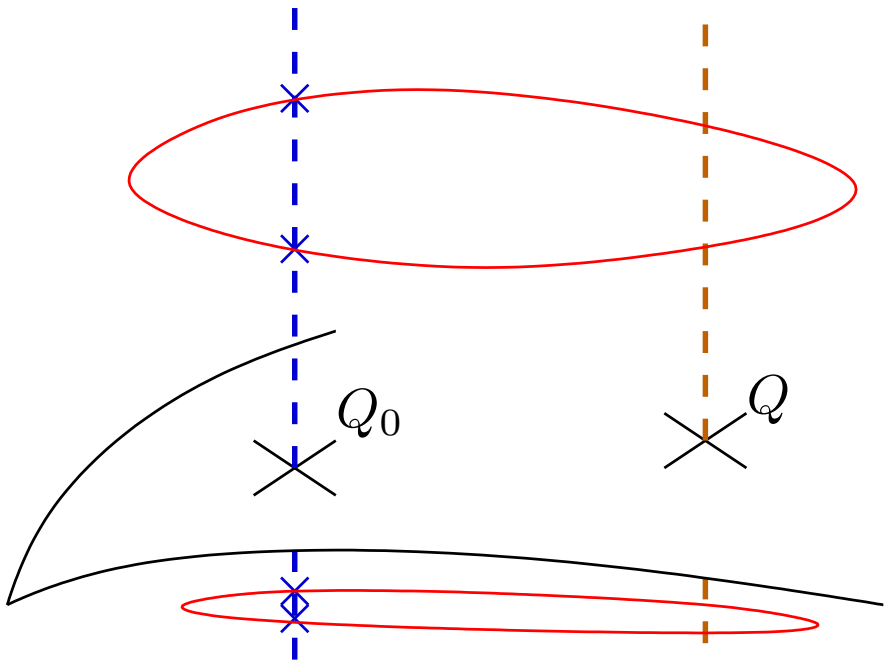


Deformation techniques

Basic idea

- The system $[3]P = Q$ is parametrized by the coordinates of Q.
- Set up a homotopy between the target $[3]P = Q$ and an initial system $[3]P_0 = Q_0$ for which we know the solutions

  basically, we let $Q_t = (1-t)Q_0 + tQ$.
- Compute a description of the solution curve and let $t = 1$.

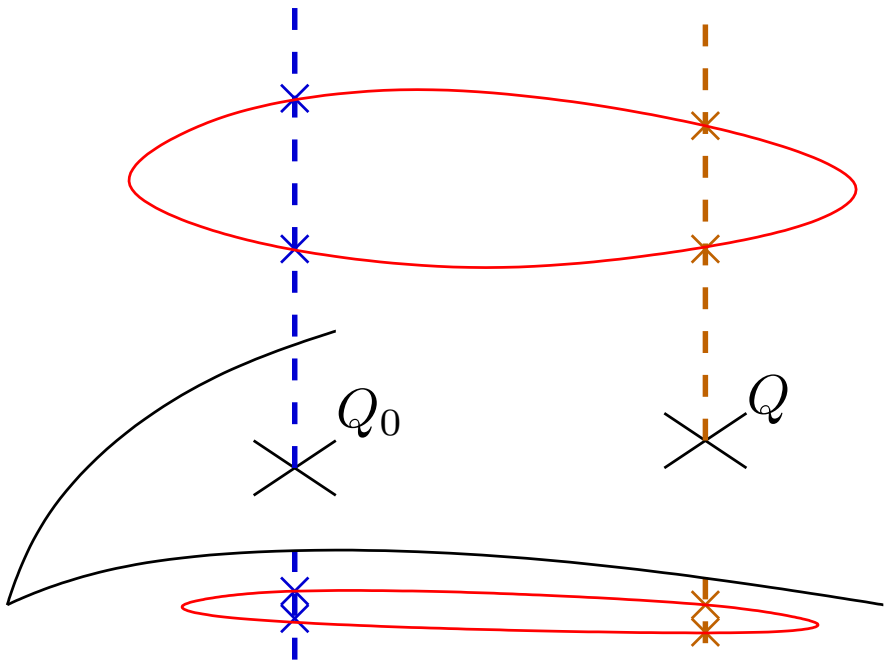


Deformation techniques

Basic idea

- The system $[3]P = Q$ is parametrized by the coordinates of Q.
- Set up a homotopy between the target $[3]P = Q$ and an initial system $[3]P_0 = Q_0$ for which we know the solutions

  basically, we let $Q_t = (1-t)Q_0 + tQ$.
- Compute a description of the solution curve and let $t = 1$.

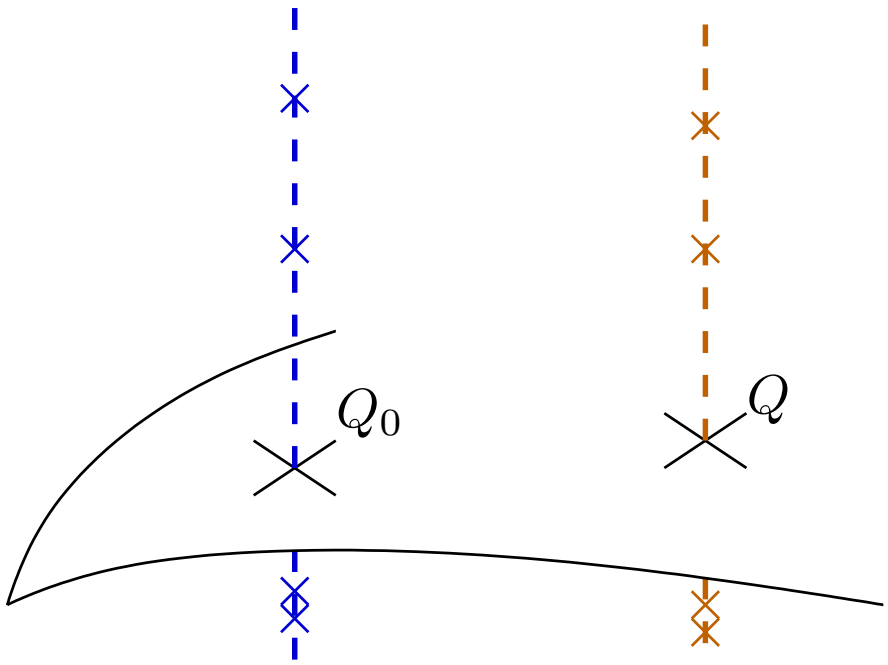


Deformation techniques

Basic idea

- The system $[3]P = Q$ is parametrized by the coordinates of Q.
- Set up a homotopy between the target $[3]P = Q$ and an initial system $[3]P_0 = Q_0$ for which we know the solutions

  basically, we let $Q_t = (1-t)Q_0 + tQ$.
- Compute a description of the solution curve and let $t = 1$.



Lifting

Main tool: Newton iteration.

1. Lifting Q. I lied:

   - We don't set $Q_t = (1-t)Q_0 + tQ$, because Q doesn't live in a linear space.
   - So we set $X(Q_t) = (1-t)X(Q_0) + tX(Q)$, and we lift the ordinates.
   - This is easy.

2. Lifting P.

Most of the time is spent evaluating the system $[3]P = Q_t$ and its Jacobian at power series.

- The system is huge: don't expand it!
- There is a "nice" straight-line program (+gradient).

The nice straight-line program

```
ZZ_pEX DT141=-tmp14c+MulTrunc(tmp14b, DT91+DT101, k)-DT121-DT131;
ZZ_pEX DT142=2*MulTrunc(tmp14b, DT61-1, k)-2*v1-DT132;
ZZ_pEX DT143=MulTrunc(-u1-1, tmp14a, k) -2*MulTrunc(tmp14b, v1, k)+T9-DT133;
ZZ_pEX DT144=tmp14a-T10;
ZZ_pEX T15=(T14-MulTrunc(T12, u1, k))/2;
ZZ_pEX DT151=(DT141-MulTrunc(DT121, u1, k)-T12)/2;
ZZ_pEX DT152=DT142/2-u1v1;
ZZ_pEX DT153=(DT143+MulTrunc(T9, u1, k))/2;
ZZ_pEX T16=(T13-MulTrunc(T12, u0, k))/2;
ZZ_pEX DT161=(DT131-MulTrunc(DT121, u0, k))/2;
ZZ_pEX DT162=(DT132-T12)/2-u0v1;
ZZ_pEX DT163=(DT133+MulTrunc(T9, u0, k))/2;
ZZ_pEX T17=-MulTrunc(DT61, T4, k)-2*MulTrunc(T15, v1, k);
ZZ_pEX DT171=MulTrunc(DT61, T3, k)-2*(T4+MulTrunc(DT151, v1, k));
ZZ_pEX DT172=-MulTrunc(DT61, T1, k)-2*MulTrunc(DT152, v1, k);
ZZ_pEX DT173=-MulTrunc(DT61, DT43, k)-2*(MulTrunc(DT153, v1, k)+T15);
ZZ_pEX DT174=-MulTrunc(DT61, DT44, k)-tmp14c+tmp13a;
ZZ_pEX T18=SqrTrunc(T15, k);
ZZ_pEX DT181=2*MulTrunc(T15, DT151, k);
ZZ_pEX DT182=2*MulTrunc(T15, DT152, k);
ZZ_pEX DT183=2*MulTrunc(T15, DT153, k);
ZZ_pEX DT184=MulTrunc(T15, DT144, k);
ZZ_pEX T19=SqrTrunc(T16, k);
ZZ_pEX DT191=2*MulTrunc(T16, DT161, k);
ZZ_pEX DT192=2*MulTrunc(T16, DT162, k);
ZZ_pEX DT193=2*MulTrunc(T16, DT163, k);
```

```
ZZ_pEX DT194=MulTrunc(T16, T10, k);
ZZ_pEX tmp20a=T15+T16;
ZZ_pEX T20=SqrTrunc(tmp20a, k)-T18-T19;
ZZ_pEX DT201=2*MulTrunc(tmp20a, DT151+DT161, k)-DT181-DT191;
ZZ_pEX DT202=2*MulTrunc(tmp20a, DT152+DT162, k)-DT182-DT192;
ZZ_pEX DT203=2*MulTrunc(tmp20a, DT153+DT163, k)-DT183-DT193;
ZZ_pEX DT204=MulTrunc(tmp20a, DT144+T10, k)-DT184-DT194;
ZZ_pEX T21=T20-SqrTrunc(T4, k);
ZZ_pEX DT211=DT201+2*MulTrunc(T4, T3, k);
ZZ_pEX DT212=DT202-2*MulTrunc(T4, T1, k);
ZZ_pEX DT213=DT203-2*MulTrunc(T4, DT43, k);
ZZ_pEX DT214=DT204-2*MulTrunc(T4, DT44, k);
ZZ_pEX T22=T19-MulTrunc(T17, T4, k);
ZZ_pEX DT221=DT191+MulTrunc(T17, T3, k)-MulTrunc(T4, DT171, k);
ZZ_pEX DT222=DT192-MulTrunc(T17, T1, k)-MulTrunc(T4, DT172, k);
ZZ_pEX DT223=DT193-MulTrunc(T17, DT43, k)-MulTrunc(T4, DT173, k);
ZZ_pEX DT224=DT194-MulTrunc(T17, DT44, k)-MulTrunc(T4, DT174, k);
ZZ_pEX T23=u1*Eu1;
ZZ_pEX T24 =u0 - T23 + Eu1Eu1 - Eu0;
ZZ_pEX tmp25=T24-Eu0;
ZZ_pEX tmp25b=Eu0*u1;
ZZ_pEX T25=MulTrunc(u0, tmp25, k) + Eu0*(SqrTrunc(u1, k)-T23+Eu0);
ZZ_pEX DT251=-u0*Eu1+2*tmp25b-Eu0Eu1;
ZZ_pEX DT252=tmp25+u0;
ZZ_pEX T26=Ev1+v1;
ZZ_pEX T27=Ev0+v0;
ZZ_pEX T28=ff4-u1;
```

Using Galoisian properties

Galois:

- there are many (81) curve branches to lift;
- but they are all conjugate:
  - if $[3]P = Q$ and $[3]P' = 0$
  - then $[3](P + P') = Q$.

So after computing the 3-torsion, we can

- lift a single curve branche;
- and add all the 3-torsion points to it
  - this is addition in the Jacobian,
  - with power series coordinates.

Interpolation

Plain.

- Knowing the 81 branches, one can recover a description of the solutions by interpolation (with rational function coefficients)

$$\begin{cases} U_3 = A_3(t, U_0) \\ U_2 = A_2(t, U_0) \\ U_1 = A_1(t, U_0) \\ Q(t, U_0) = 0, \end{cases} \qquad \deg(Q) = 81.$$

- Use rational interpolation.

$$\begin{cases} U_3 = A_3^\star/Q' \\ U_2 = A_2^\star/Q' \\ U_1 = A_1^\star/Q' \\ Q = 0, \end{cases} \qquad \deg(Q) = 81.$$

- Then, factor Q.

Triangular interpolation

Using the 3-torsion action.

- Let G_{27} be a subgroup of size 27 of the 3-torsion.
- The 81 branches group into 3 orbits, so the orbit-sum O of U_0 satisfies

$$R(t,O)=0, \quad R'(t,O,U_0)=0, \qquad \deg(R)=3, \quad \deg(R')=27.$$

Continue with subgroups: we get

$$\begin{cases} R'''(t,O,O',O'',U_0) \\ R''(t,O,O',O'') \\ R'(t,O,O') \\ R(t,O), \end{cases}$$

where all polynomials have degree 3 (we still use rational interpolation).

Experiments

Experiments

Cuvre defined over $\mathbb{F}_p$ with $p = 2^{127} - 1$ by the equation:

$$\begin{aligned} y^2 = \; & 313753763479717340856704966098366157 26 + \\ & 279536050382142212536459814755115706 57x + \\ & 624200036268498524283325544377652771 61x^2 + \\ & 880059549395273872442398492840584736 79x^3 + \\ & 155062477294917469622604436777931982527x^4 + x^5. \end{aligned}$$

Its Kummer surface has parameters:

$$\begin{aligned} a &= 707492735370197151974876966608573181 00 \\ b &= 132971112936989975185308056294930534 56 \\ c &= 319627247886293737203483622555158934 54 \\ d &= 399618462052043836086943134607955309 17 \end{aligned}$$

Timings obtained on a few 8GB Opteron 2.4 Ghz.

Results: large ℓ

ℓ	Time for one resultant	Number of resultants	Total time
11	0.01976	49274	974
13	0.03264	97202	3173
17	0.06737	287582	19374
19	0.08707	450194	39198
23	0.15532	970742	150776
29	0.24886	2461634	612602
31	0.27200	3216494	874886

2004: $\ell = 19$.

Results: large ℓ

ℓ	Cleaning	Frobenius	Schoof
11	411	148	45
13	1174	312	203
17	3685	1183	927
19	12443	3382	2245
23	40729	14581	12725
29	134993	31456	37285
31	178431	38213	44493

Results: 2^k-torsion

Torsion	Degree ext.	Time halving	Time Schoof
2^{10}	512	20.70	12
2^{11}	1024	70	30
2^{12}	2048	243	72
2^{13}	4096	903.8	167
2^{14}	8192	3554	425
2^{15}	16384	13329	1026
2^{16}	32768	58751	2753
2^{17}	65536	257425	9842

2004: 2^{10} torsion.

Results: 3^k-torsion

Torsion	Degree	Halving	Schoof
3^2	6	543	0.1
3^3	6	57	0.1
3^4	18	59	0.2
3^5	54	265	1
3^6	162	1330	4
3^7	486	4905	11
3^8	1458	22642	44
3^9	4374	134530	211

2004: 27-torsion.

Finally

We got (s_1, s_2) modulo

$$m = 2^{14} \times 3^8 \times 5^3 \times 7^2 \times 11 \times 13 \times 17 \times 19 \times 23 \times 29 \times 31.$$

The final step takes about 2h.

Next

A large scale computation

- about one month per curve
- early abort strategies
- probably 2000 to 3000 curves to try.