Point counting in genus 2: towards 128 bits

P. Gaudry	É. Schost	N. Pitcher
Cacao project	ORCCA	MSCS
CNRS-INRIA	UWO	U. of Illinois

Genus 2 curves and associated objects

- 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
  - projective variety of dimension 2;
  - but we will work on an affine part of it.

- $\mathbf{K}$ is the associated Kummer surface
  - $\mathbf{K} = \mathbf{J}/\{\pm 1\}$.
  - a projective variety of dimension 2 too;
  - we won't work with it too much.

Why?

Cryptologists like genus 2 Jacobians.

- Dimension 2:
  - there are more rational points than on an elliptic curve,
  - so for a given number of points, smaller base field.
- Scalar multiplication is not too slow
  - Chudnovsky^2, Gaudry.

So they could be competitive with elliptic curves.

Our question

Problem: finding one curve

- whose Jacobian has a prime cardinality (for safety reasons);
- over a prime field;
- with small coefficients
  - to make scalar multiplication fast.

Goal: $p = 2^{127} - 1$.

We are not there yet, but almost.

- a first 128 bit run;
- the curve was mostly random (but lucky).

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, $p \simeq 2^{61}$.
- Matsuo-Chao-Tsujii (2002): baby steps / giant steps strategy.
- Gaudry-S. (2004): cryptographic size: $p \simeq 2^{82}$.
- Gaudry-S. (2004): parallel, low-memory version of Matsuo-Chao-Tsujii.

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.

To keep in mind:

- 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?)

- solving bivariate systems by bivariate resultants.

Prime powers:

- cool improvements on 2^k-torsion and 3^k-torsion;
- brute-force 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
  - ./configure ; make
- On the other hand, no Gröbner engine
  - anyway, faster workarounds.

Reduction to bivariate solving

Almost everything is 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!
- Solve bivariate equations with bivariate resultants.

What's left to improve:

- Bivariate resultants are sub-optimal.

  Output size $\simeq \ell^4$; cost $O\tilde{}(\ell^6)$

  $O\tilde{}$ means we neglect logarithmic factors.
- Systems are over-determined.

Lifting the torsion

`While (possible==true) do`

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

Here, $\ell = 2, 3, 5, 7$.

- Known complexity estimates ill-adapted: it's all in the big-Oh.
- 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.

Division by 2

1. Amounts to compute square roots.

- ...or a little bit more (extend the base field when no square root exists)
- complexity as in Kaltofen-Shoup, 1997.

2. 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
  - equal-degree factorization
  - finding new primitive elements
- cost: $C(d) = O\tilde{}(d^{1.5})$ (polynomial operations) $+ O(d^2)$ (linear algebra)
- Kedlaya-Umans: not useful yet.

Division by 3

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

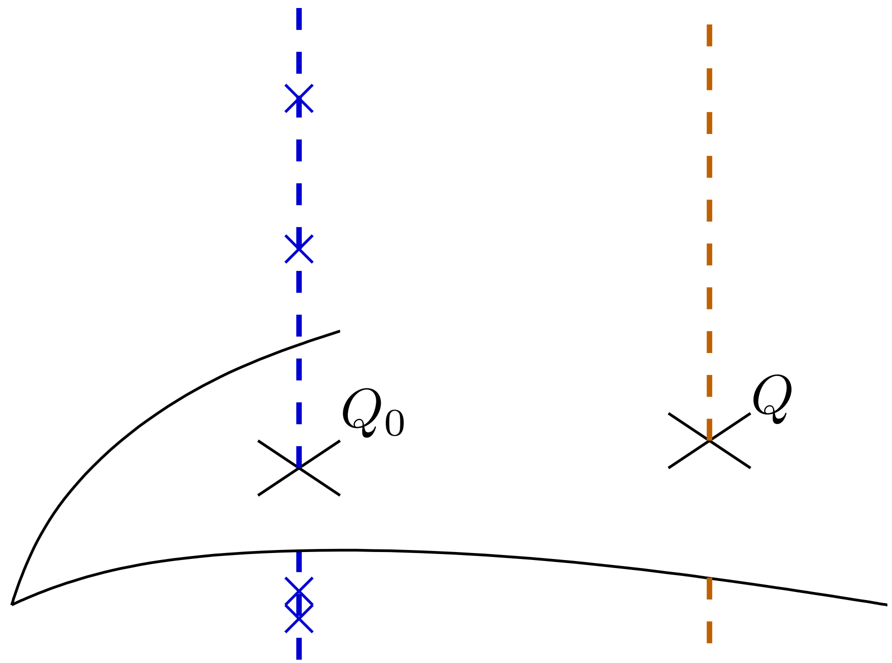
  basically, we let $Q_t = (1-t)Q_0 + tQ$.
- Compute a description of the solution curve and let $t = 1$.

Division by 3

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 $t = 1$.

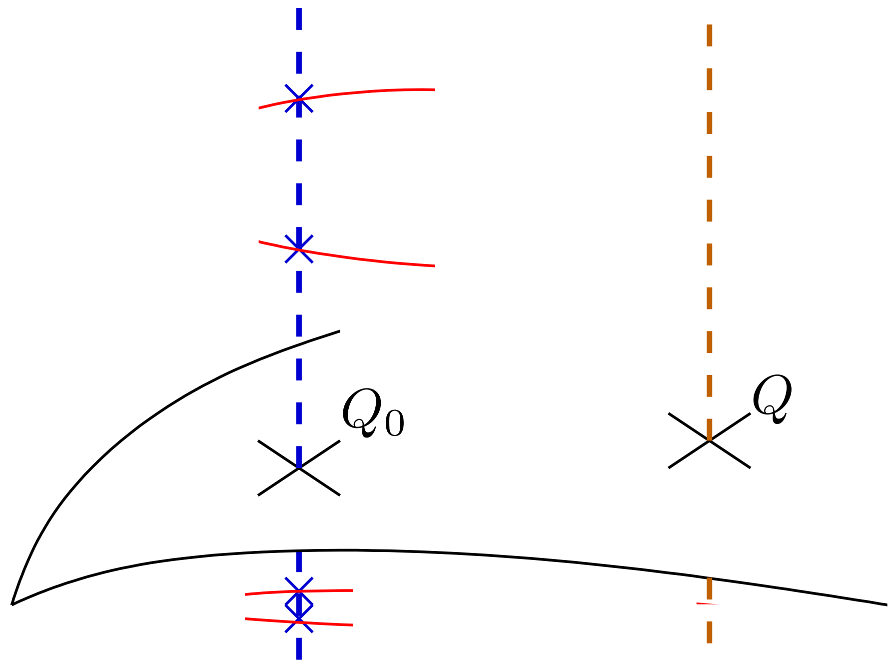


Division by 3

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 $t = 1$.

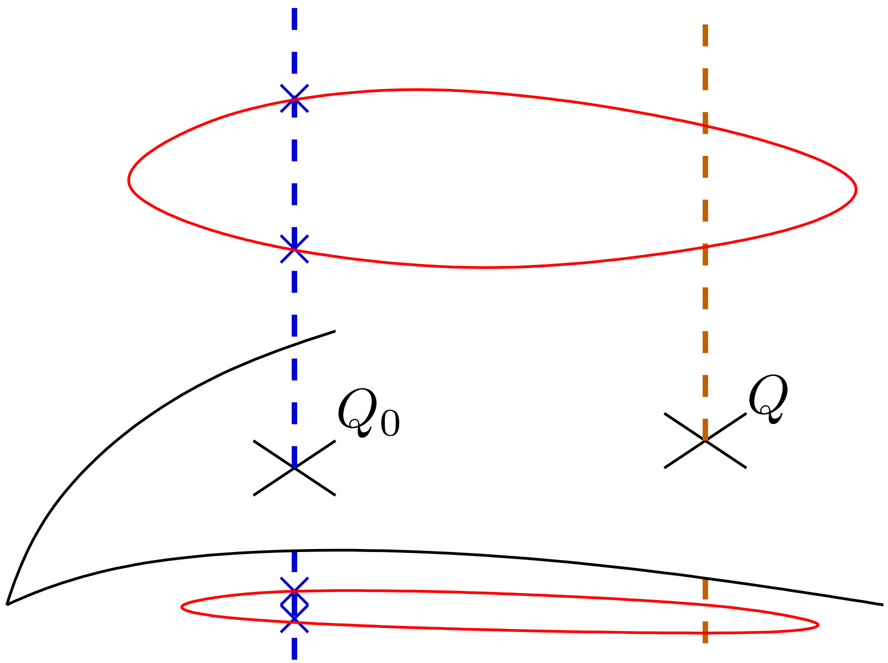


Division by 3

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 $t = 1$.

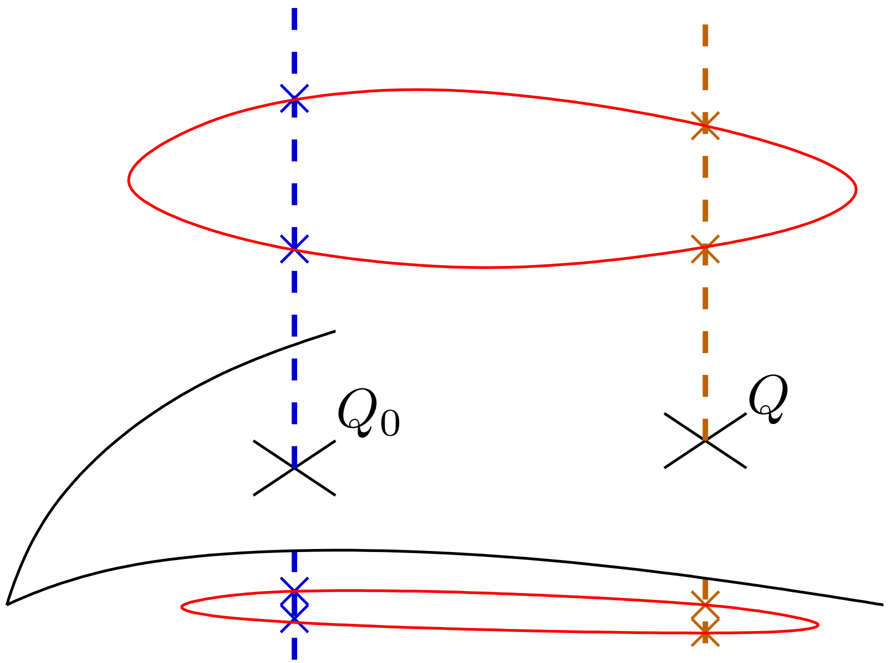


Division by 3

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 $t = 1$.

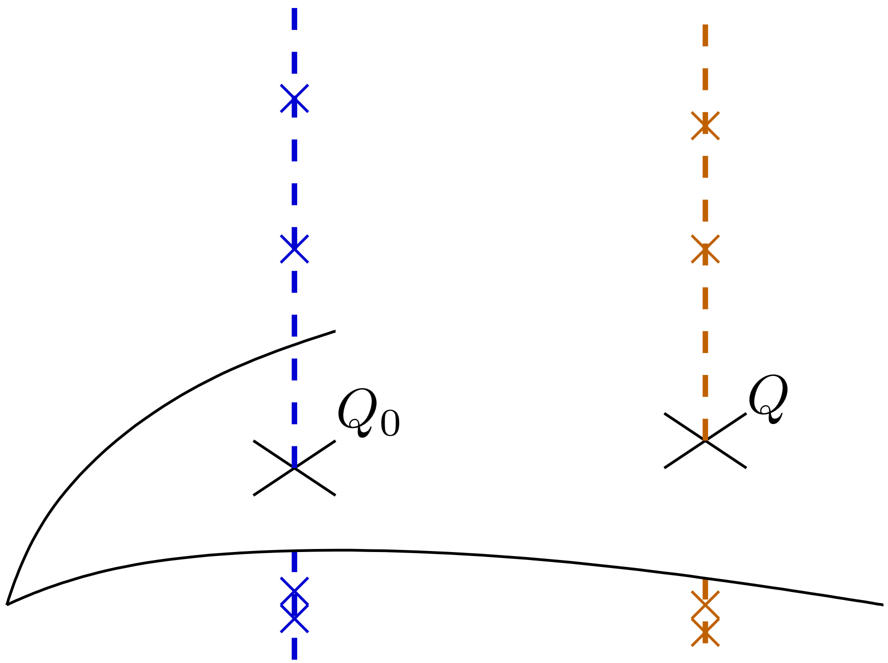


Division by 3

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 $t = 1$.



Concretely

General idea: as in Pardo-San Martin; Matera et al.

1. Evaluation properties

- nice straight-line program for multiplication by 3.

2. Action of the 3-torsion

- branches are conjugate: lift only one branch.

3. Local $\rightarrow$ global

- rational interpolation.

4. Subgroups of the 3-torsion

- easier factorization.

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;
```

Results

task	time in sec.	bottleneck
$\ell = 2^{17}$	348,008	modular composition
$\ell = 3^7$	126,720	lifting / interpolation
		`polynomial too large for FFT`
$\ell = 5^4$	235,713	bivariate resultants
$\ell = 7^2$	95,975	bivariate resultants
$\ell = 11, \ldots, 31$	2,242,185	bivariate resultants
		memory becomes a problem
kangaroos	7300	
total	$3{,}055{,}901 \approx 1$ month	

We expect to test about 10000 curves before finding a suitable one.

Should take about 100 CPU years.