Searching a secure genus 2 curve with small parameters over a prime field

Pierrick Gaudry and Éric Schost

pierrick.gaudry@loria.fr

CNRS – INRIA – Nancy Université

University of Western Ontario

Contents

- Hyperelliptic curves: definitions and basic properties
- The point counting problem
- Speed-driven choices
- Algorithm used to count points
- In practice
- Perspectives

Hyperelliptic curves

Definition

Def. A hyperelliptic curve $\mathcal{C}$ is a curve of equation

$$y^2 = f(x),$$

with f a squarefree, monic polynomial of odd degree $2g+1$, where g is the genus of $\mathcal{C}$.

NB: in char. 2, change this to $y^2 + h(x)y = f(x)$.

The Jacobian of $\mathcal{C}$ is a variety of dimension g, that has a natural group law

$\Longrightarrow$ Abelian variety.

Can be described in terms of classes of degree 0 divisors of $\mathcal{C}$.

Mumford representation

Consequence of Riemann-Roch theorem:

Prop. Each element of $\mathrm{Jac}(\mathcal{C})$ can be uniquely represented by a divisor of the form

$$D = P_1 + P_2 + \cdots + P_r - r\infty,$$

with $r \leq g$ and P_i is not symmetric of P_j.

Rem. D is defined over $\mathbb{F}_q$ if and only if the (P_i) are defined over $\mathbb{F}_q$ as a whole. They can be individually defined over an extension.

Prop. Each element of $\mathrm{Jac}(\mathcal{C})$ can be uniquely represented by a pair of polynomials $\langle u(x), v(x)\rangle$, with $\deg v < \deg u \leq g$ and $u|v^2 - f$.

Rem. : if $P_i = (x_i, y_i)$, then $u(x) = \prod(x - x_i)$ and v is such that $v(x_i) = y_i$.

Mumford representation (2)

Prop. $\langle u(x), v(x) \rangle$ represents an $\mathbb{F}_q$-rational element iff u and v have coefficients in $\mathbb{F}_q$.

Group law can be efficiently computed with Mumford representations:

Cantor's algorithm

- A few operations with polynomials of degree up to $2g$;
- Naïve complexity: $O(g^3)$;
- Asympotitically fast variant: $\tilde{O}(g)$;
- For fixed small genus: unroll/optimize, explicit formulas.

Cardinalities

If the base field is $\mathbb{F}_q$, the set of points of $\mathcal{C}$ and $\mathrm{Jac}(\mathcal{C})$ are finite.

Thm. (Weil conjectures)

$$|\#\mathcal{C} - (q+1)| \leq 2g\sqrt{q}$$

$$(\sqrt{q}-1)^{2g} \leq \#\mathrm{Jac}(\mathcal{C}) \leq (\sqrt{q}+1)^{2g}$$

i.e $\#G$ is about q^g.

Rem. If one wants a group order of (say) 256 bits, the base field can be smaller when one takes a higher genus curve.

The point counting problem

Point counting

Three classes of algorithms.

- Schoof's method.
  - 1985 for elliptic curves.
  - polynomial time for fixed genus.
  - use ℓ-torsion for several small ℓ.
- Satoh's algorithm.
  - 2000.
  - use a p-adic canonical lifting.
  - polynomial time in small characteristic and fixed genus.

Point counting

- Kedlaya's and Lauder-Wan's algorithms.
  - 2001.
  - use a p-adic lifting + cohomology.
  - polynomial time in small characteristic.
  - can be combined with deformation techniques:
    - precompute data for a family of curves
    - each curve can then be counted quickly
    - see works of Castryck, Gerkmann, Hubrechts, Vercauteren, ...

Rem. The p-adic algorithms have proven to scale pretty well with genus.

Rem. Kedlaya's algorithm scales not so badly with p (Harvey's algorithm).

Genus 2 over prime fields

Timeline:

- G.-Harley (2000): 126 bit Jacobian;
- G.-Schost (2004): 164 bit, secure Jacobian;
- Sutherland (2007): 188 bit, secure Jacobian;
- G.-Schost (2008): 256 bit Jacobian.
- G.-Schost (today): 256 bit, doubly-secure Jacobian.

Speed-driven choices

Inspired by Bernstein's `curve25519`

Choosing the base field

Prime field:

- faster than small characteristic in software (but AVX?);
- considered more secure by some people (inc. NSA?).

We want an AES-128 equivalent security $\Longrightarrow$ p must have about 128 bits.

For maximal efficiency, p should be Mersenne-like.

It's just too tempting to sacrifice one bit and take

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

Reduction modulo p is cheap (a few shifts and adds).

Allow Kummer surface arithmetic

The Kummer surface of a genus 2 curve is the analogous of x-only formulae for elliptic curves:

$$K(\mathcal{C}) = \mathrm{Jac}(\mathcal{C})/\{-1, 1\}.$$

Always well-defined.

To get maximum speed, some rationality conditions must be added (much akin to Montgomery form):

- Squares of Theta constants must be rational;
- This implies all 2-torsion elements to be rational;
- Forces a 16 factor in group order;

Small parameters

Group law (doubling) looks like:

1. $x' = A^2(x+y+z+t)^2$;
2. $y' = B^2(x+y-z-t)^2$;
3. $z' = C^2(x-y+z-t)^2$;
4. $t' = D^2(x-y-z+t)^2$;
5. $X = a^2(x'+y'+z'+t')^2$;
6. $Y = b^2(x'+y'-z'-t')^2$;
7. $Z = c^2(x'-y'+z'-t')^2$;
8. $T = d^2(x'-y'-z'+t')^2$;
9. Return $2P = (X, Y, Z, T)$.

where $a^2, b^2,, \ldots, D^2$ are parameters depending on the curve.

Taking all of them small (tiny constants) speeds up computation.

NB: not possible with CM construction; need point counting.

Doubly secure

We ask that:

- Group order of Jacobian of $\mathcal{C}$ is 16 times a prime;
- Group order of Jacobian of twist of $\mathcal{C}$ is 16 times a prime.

This saves some possibly annoying point validation phase: an element of the Kummer surface can correspond to an element of the Jacobian of the curve or of its twist.

If the twist is not secure, before multiplying the point by our secret, one must check that we are not on the twist.

Also, killing cofactors is as easy as operating one additional doubling (the 16 is a $(\mathbb{Z}/2\mathbb{Z})^4$ subgroup).

Algorithm used to count points

Schoof's algorithm – torsion

Let $\mathcal{C}$ be a curve of genus g over $\mathbb{F}_q$.

Schoof's algorithm relies on torsion elements.

Thm. Let ℓ be coprime to the characteristic. Then

$$\mathrm{Jac}(\mathcal{C})[\ell] \cong (\mathbb{Z}/\ell\mathbb{Z})^{2g},$$

(taking points over algebraic closure).

Rem. This variety can be described by an ideal I_ℓ, the ℓ-torsion ideal.

Ex. In genus 1, I_ℓ is essentially the division polyomial ψ_ℓ of the elliptic curve.

Schoof's algorithm – Frobenius

The Frobenius endomorphism π is defined by raising all the coordinates to the q-th power.

Thm. π verifies a polynomial equation $\chi(\pi) = 0$ of the form

$$\chi(t) = t^{2g} - s_1 t^{2g-1} + \cdots + (-1)^g s_g t^g + (-1)^{g+1} s_{g-1} t^{g-1} + \cdots - s_1 q^{g-1} t + q^g,$$

where $s_1, \ldots, s_g$ are integers such that all the roots of χ have absolute value $\sqrt{q}$.

Rem. π fixes elements that are defined over $\mathbb{F}_q$.

Cor. $\#\mathrm{Jac}(\mathcal{C}) = \chi(1)$.

Schoof's algorithm

- While not enough modular information do:
  - Choose next small prime ℓ;
  - Compute I_ℓ;
  - Check for which values of $s_1, \ldots, s_g$ mod ℓ we have

$$\chi(\pi)(D) = 0,$$

    for D a generic ℓ-torsion element.
  - If there is only one g-uple that fits, then we known $\chi(t) \mod \ell$.
- Reconstruct $\chi(t)$ by CRT and Weil's bound on coeffs.

Main difficulty: computing I_ℓ. Cost = $\tilde{O}(\ell^6)$ operations in genus 2.

Helping with exponential-time algorithms

Division by small primes:

- From an ℓ^k-torsion element D_k, get an ℓ^{k+1}-torsion element D_{k+1};
- Division done by solving algebraic system;
- Field of definition for D_k grows quickly.

Final birthday paradox computation:

- When the quantity of remaining choices for $\chi(t)$ is small enough, use a $\sqrt{\ }$ algorithm.
- If s_1 is completely determined, use kangaroos for finding s_2;
- Otherwise, use a bidimensional random walk (cockraoches).

  Improved recently by Galbraith-Ruprai.

A few ugly details in ℓ-torsion ideal computation

Let I_ℓ be the ℓ-torsion ideal. It is obtained as the solutions of

$$[\ell] \left\langle X^2 + u_1 X + u_0, v_1 X + v_0 \right\rangle = 0.$$

In order to compute modulo I_ℓ, one needs a lex-ordered Gröbner basis (other orders do not permit fast arithmetic).

Due to the form of the equations, it is possible to get R and S such that

$$R(u_1) \quad \text{and} \quad u_0 - S(u_1)$$

are in I_ℓ, at the cost of one bivariate resultant computation.

- How to compute R and S quickly ?
- How to clean them and obtain the exact ideal I_ℓ?

Computing a bivariate resultant

Let $f(x, y)$ and $g(x, y)$ be two bivariate polynomials. One wants $\mathrm{Res}_y(f, g)(x)$.

The fastest known method is evaluation / interpolation:

1. Compute $f_i(y) = f(x_i, y)$ and $g_i(y) = g(x_i, y)$ for many values x_i;
2. Compute $R_i = \mathrm{Res}(f_i, g_i)$ for all i;
3. Interpolate $R(x)$ from its value R_i at x_i.

Remarks:

- Due to the form of f and g, they can be evaluated in $O(\ell^2)$ operations;
- When computing R_i, compute also the last subresultant: this give S;
- Choose the x_i in a geometric progression: save a log factor.

Computing a bivariate resultant

Complexity:

Most of (all) the time is spent in computing individual resultants.

Degree of R and S is in $7/2\ \ell^4$.

$\Longrightarrow$ Cost is $O(\ell^4 M(\ell^2) \log \ell)$ operations in $\mathbb{F}_p$.

Rem. For large values of ℓ, we use recursive algorithm for resultant (well implemented in NTL).

Rem. The costly part is simply, higly parallelizable, and requires low memory.

Cleaning R and S

R is a multiple of the smallest polynomial in u_1 that lies in I_ℓ. Solutions to clean it:

- Compute another resultant. Double the time.
- Plug the result in defining polynomials: faster, but requires $O(\ell^5)$ memory.
- Use the group law! Cost $O(M(\ell^4)\log\ell)$.

Work in the algebra $A = (\mathbb{F}_p[x]/R(x))[T]/(T^2 - xT + S(x))$.

Let $P = \langle X - T, \sqrt{f(-T)}\rangle$ and $Q = \langle X - (x - T), \sqrt{f(x - T)}\rangle$.

If R was clean, $[\ell]P$ would be opposite of $[\ell]Q$.

$\Longrightarrow$ compute the difference of abscissae of these multiples, take GCD with R, this gives a new clean R.

In practice

Preselecting curves

- Create a list of curves with small parameters; we took a^2, b^2, c^2, d^2 in $[0, 40]$;
- Eliminate duplicates due to isomorphic curves;
- Compute group orders modulo $32, 3, 5, 7$, and eliminate those for which we can not be doubly good.

This step is very cheap and is done on a single processor in a few hours.

Computing modulo $\ell = 11, 13, \ldots, 31$

We give data for $\ell = 31$, the largest we used.

Computing individual resultants is done in parallel. We need a bit more than 3.2 millions of them.

On a single core, this would take 10 days (no RAM).

Interpolation of the resultant takes about 2 hours.

Cleaning the resultant and reconstructing the ideal takes 1 day.

Testing characteristic polynomial of Frobenius takes 1 day.

These 3 final steps are sequential and require 4 GB of RAM.

Computing modulo powers

For 2^k computation, we halved a 4-torsion divisor until we reached an extension of degree 65536.

On average, this gives (s_1, s_2) modulo $(16384, 65536)$.

For 3^k, the division by 3 is computed using a deformation technique. (on average modulo $3^{6.8}$)

For 5^k and 7^k, the system is solved using evaluation/interpolation and Newton lifting. (5^3 and 7^2 at best)

Most of this is sequential and memory consuming!

Total time

We abort ASAP: compute mod ℓ up to 31 before compute modulo powers.

For a fully counted curve: about 6 CPU-weeks in total.

About 1.5 million CPU-hours spent – 6 months of clock time.

Ressources: Sharcnet's grid computing facility

- `whale` cluster: throughput cluster
  - 768 nodes, Opteron at 2.2 GHz
  - 4 cores per node
  - 4 GB RAM per node

  Well suited for computing individual resultants.
- `bull` cluster:
  - 96 nodes, Opteron at 2.4 GHz
  - 4 cores per node
  - 32 GB RAM per node

  Useful for powers and finishing largest ℓ.

The curve

We completely counted 584 curves and got 50 semi-good curves.

Among them, one is doubly-good: $\mathcal{C}_{11,-22,-19,-3}$

Its Weierstrass equation is

$$y^2 = \texttt{3d74bd441c0152afbc7cf8075391f7fd} + \texttt{76a252045e85803cc14250f839d08a6}\,x + \texttt{7468e5316ea9437dba3769f4b2d902eb}\,x^2 + \texttt{39a7ca7f0f13f73f29b62076ff8a8e1a}\,x^3 + \texttt{3074a29f6a57649a1edfa9379ae1f0db}\,x^4 + x^5$$

and the characteristic polynomial is $T^4 - s_1T^3 + s_2T^2 - s_1pT + p^2$ with

$s_1 = -\texttt{669ad30418ea5298}$ and $s_2 = -\texttt{2c27fc9f9a154ff47730b4b840c05bd2}$.

One gets the group orders:

$2^4 \times \texttt{4000000000000000334d69820c75294ad3d8036065eab00b88cf4b47bf3fa43}$

$2^4 \times \texttt{3fffffffffffffffccb2967df38ad6b2d3d8036065eab00b88cf4b47bf3fa43}$

Perspectives

Speed???

Not yet implemented!

With Edwards's model, it's getting hard to beat elliptic curves.

It will depend on the practical costs of

- multiplications by small constants;
- squares vs multiplications.

Further point counting?

With our current implementation, and current technology:

- Possible to build a semi-good curve of security 2^{160};
- Possible to count one curve of size corresponding to security 2^{192}.
- Not possible to reach security 2^{256}.

Rem. Recent progresses in explicit isogeny computations (Smith, Lubicz-Robert) open new perspectives for Elkies-Atkin tricks in genus 2.