Construction of Secure Random Curves of Genus 2 over Prime Fields

Pierrick Gaudry[1] and Éric Schost[2]

[1] Laboratoire LIX, École polytechnique, France
gaudry@lix.polytechnique.fr
[2] Laboratoire STIX, École polytechnique, France
Eric.Schost@polytechnique.fr

Abstract. For counting points of Jacobians of genus 2 curves defined over large prime fields, the best known method is a variant of Schoof's algorithm. We present several improvements on the algorithms described by Gaudry and Harley in 2000. In particular we rebuild the symmetry that had been broken by the use of Cantor's division polynomials and design a faster division by 2 and a division by 3. Combined with the algorithm by Matsuo, Chao and Tsujii, our implementation can count the points on a Jacobian of size 164 bits within about one week on a PC.

1 Introduction

Genus 2 hyperelliptic curves provide an interesting alternative to elliptic curves for the design of discrete-log based cryptosystems. Indeed, for a similar security, the key or signature lengths are the same as for elliptic curves and furthermore the size of the base field in which the computations take place is twice smaller. During the last years, efforts in improving the group law algorithms made these cryptosystems quite competitive [19, 25].

To ensure the security of the system, it is required to have a group of large prime order. Until recently, for the Jacobian of a genus 2 curve, only specific constructions provided curves with known Jacobian order, namely the complex multiplication (CM) method [34] and the Koblitz curves. These curves have a very special structure; although nobody knows if they are weaker than general curves, it is pertinent to consider random curves as well. This raises the problem of point-counting: given a random curve, find the group order of its Jacobian.

With today's state of the art, the complexity of the point counting task in genus 2 highly depends on the size of the characteristic of the base field: in short, the smaller the characteristic, the easier the task of point counting ("easy" means fast and does not mean that the theoretical tools are simple).

In the case of genus 2 curves in small characteristic p, the point counting problem was recently solved using p-adic methods [31, 23, 20]. The particular case where $p = 2$ is in fact treated almost as quickly as in genus 1. Unfortunately, these dramatic improvements do not apply when p becomes too large (say, a few thousands [10]).

For large p, the best known algorithms are variants of Schoof's algorithm, theoretical descriptions of which can be found in [26, 18, 1, 16]. In 2000, Gaudry and Harley [11] designed and implemented the first practical genus 2 Schoof algorithm, making use of Cantor's division polynomials [8]. To reach reasonable sizes, however, it was necessary to combine the Schoof approach with a Pollard lambda method. Their record was a random genus 2 curve over a prime field of size about 10^{19}, thus too small to be used in a cryptosystem. For "medium characteristic", they also proposed to use the Cartier-Manin operator to get additional information that can be combined with others. Therefore, for medium characteristic p (say 10^9, see [5]), point counting is easier than for very large p.

We mentioned that in the non-small characteristic case, once the group order has been computed modulo some large integer, the computation is finished using a Pollard lambda method. For this last phase, Matsuo, Chao and Tsujii [21] proposed a Baby-step/Giant-step algorithm that speeds up this phase. With this device and using the Cartier-Manin trick, they performed a point counting computation of cryptographical size for a medium characteristic field.

In this paper, we improve on the methods of [11], so that, combined with the algorithm of [21], we can reach cryptographical size over prime fields. Our improvements are concerned with the construction and the manipulation of torsion elements in the Schoof-like algorithm of [11]. The impact of these improvements is asymptotically by a constant factor, but they yield significant speed-up in practice for the size of interest in cryptography. We now summarize them:

Our first contribution is the reintroduction of symmetries that were lost in [11]. Indeed, the use of Cantor's division polynomials to construct torsion elements is very efficient, but the resulting divisor is given as a sum of points instead of in Mumford representation. Therefore a factor of 2 in the degrees of the polynomials that are manipulated is lost. In Sections 3.2 and 3.3, we give algorithms to save this factor of 2 in the degrees.

In [11], it is proposed to build 2^k-torsion elements using a halving algorithm based on Gröbner basis computations. Our second contribution is a faster division by 2, using a better representation of the system; in the same spirit we show that a division by 3 can also be done: this is described in Section 4. Another practical improvement is the ubiquitous use of an explicit action on the roots coming from the group law to speed-up the factorizations that occur at different stages. We explain it in details in the case of the division by 2 in Section 3.4.

To illustrate and to test the performance of our improvements, we implemented them in Magma or NTL and mixed them with the algorithm of [21] and an early abort strategy. Our main outcome is the first construction of secure random curves of genus 2 over a prime field, as we obtained Jacobians of prime order of size about 2^{164}.

2 Generalities

In this work, p denotes a fixed odd prime, $\mathbb{F}_p$ is the finite field with p elements, and $\mathcal{C}$ is a genus 2 curve defined by the equation $y^2 = f(x)$, where f is a

squarefree monic polynomial in $\mathbb{F}_p[X]$ of degree 5. The main object we consider is the Jacobian $\mathbf{J}(\mathcal{C})$ of $\mathcal{C}$. We handle elements of $\mathbf{J}(\mathcal{C})$ through their Mumford representation: each element of $\mathbf{J}(\mathcal{C})$ can be uniquely represented by a pair of polynomials $\langle u(x), v(x)\rangle$, where u is monic of degree at most 2, v is of degree less than the degree of u, and u divides $v^2 - f$. The degree of the u-polynomial in Mumford's representation is called the *weight* of a divisor. If K is an extension field of $\mathbb{F}_p$, we may distinguish the curves defined on K and $\mathbb{F}_p$, by denoting them $\mathcal{C}/K$ and $\mathcal{C}/\mathbb{F}_p$; the Jacobians are correspondingly denoted by $\mathbf{J}(\mathcal{C}/K)$ and $\mathbf{J}(\mathcal{C}/\mathbb{F}_p)$. For precise definitions and algorithms for the group law, we refer to [22] and [7, 19].

Let $\overline{\mathbb{F}_p}$ be an algebraic closure of $\mathbb{F}_p$ and let us consider the Frobenius endomorphism on $\mathbf{J}(\mathcal{C}/\overline{\mathbb{F}_p})$ denoted by π. By Weil's theorem (see [24]), the characteristic polynomial $\chi(T)$ of π has the form $\chi(T) = T^4 - s_1T^3 + s_2T^2 - ps_1T + p^2$, where s_1 and s_2 are integers such that $|s_1| \leq 4\sqrt{p}$ and $|s_2| \leq 6p$. Furthermore $\#\mathbf{J}(\mathcal{C}) = \chi(1) = p^2 + 1 - s_1(p+1) + s_2$.

In point-counting algorithms based on Schoof's idea [27], the torsion elements of $\mathbf{J}(\mathcal{C})$ play an important role. If N is a positive integer, the subgroup of N-torsion elements of $\mathbf{J}(\mathcal{C}/\overline{\mathbb{F}_p})$ is a finite group denoted by $\mathbf{J}(\mathcal{C})[N]$; it is isomorphic to $(\mathbb{Z}/N\mathbb{Z})^4$ and has the structure of a free $\mathbb{Z}/N\mathbb{Z}$-module of dimension 4 (see [24]). Furthermore, the characteristic polynomial of the restriction of π to $\mathbf{J}(\mathcal{C})[N]$ is $\chi(T)$ mod N. Applying this to different small primes or prime powers leads to the genus 2 Schoof algorithm that is sketched in Algorithm 1.

Algorithm 1 Sketch of a genus 2 Schoof algorithm

1. For sufficiently many small primes or prime powers ℓ:
   (a) Let $L = \{(s_1, s_2);\quad s_1, s_2 \in [0, \ell - 1]\}$.
   (b) While $\#L > 1$ do
   - Construct a new ℓ-torsion divisor D;
   - Eliminate those elements (s_1, s_2) in L such that

$$\pi^4(D) - s_1\pi^3(D) + s_2\pi^2(D) - (ps_1 \bmod \ell)\pi(D) + (p^2 \bmod \ell)D \neq 0$$

   (c) Deduce $\chi(T)$ mod ℓ from the remaining pair in L.
2. Deduce $\chi(T)$ from the pairs $(\ell, \chi(T) \bmod \ell)$ by Chinese remaindering, or using the algorithm of [21].

Our contribution is to improve the first part of the algorithm, the construction of ℓ-torsion divisors; the computations for small primes and prime powers are respectively described in Sections 3 and 4.

We will frequently make genericity assumptions on the curve $\mathcal{C}$ and its torsion divisors. We assume that $\mathcal{C}$ is chosen randomly among genus 2 curves defined over a large field $\mathbb{F}_p$, so we can expect that with high probability, such assumptions are satisfied. The cases when our assumptions fail should require special treatments, which are not developed here.

For the complexity estimates, we denote by $\mathsf{M}(d)$ the number of $\mathbb{F}_p$-operations required to multiply two polynomials of degree d defined over $\mathbb{F}_p$. We make the classical assumptions on M (see for instance [32, Definition 8.26]). In the sequel,

if no precise reference is given for an algorithm, then it can be found in [32], together with a complexity analysis in terms of M.

3 Computation modulo a small prime ℓ

In the classical Schoof algorithm for elliptic curves, a formal ℓ-torsion point is used: the computations are made with a point $P = (x, y)$, where x cancels the ℓ-th division polynomial ψ_ℓ and y is linked to x by the equation of the curve. In other words, we work in a rank 2 polynomial algebra quotiented by two relations: $\mathbb{F}_p[x, y]/(\psi_\ell(x), y^2 - (x^3 + ax + b))$.

In genus 2, we imitate this strategy. According to [18], it is enough to consider the ℓ-torsion divisors of weight 2 (this is not surprising since a generic divisor has weight 2). Let thus D be a weight 2 divisor given in Mumford representation, $D = \langle x^2 + u_1 x + u_0, v_1 x + v_0 \rangle$. Then there exists a radical ideal I_ℓ of $\mathbb{F}_p[U_1, U_0, V_1, V_0]$ such that

$$D \in \mathbf{J}(\mathcal{C})[\ell] \iff \varphi(u_1, u_0, v_1, v_0) = 0, \ \forall \varphi \in I_\ell.$$

By analogy with elliptic division polynomials, this ideal I_ℓ is called the *ℓ-th division ideal*. There are $\ell^4 - 1$ non-zero ℓ-torsion elements, so that I_ℓ has dimension 0 and degree at most $\ell^4 - 1$; generically, by the Manin-Mumford conjecture [15, p. 435], all non-zero torsion divisors have weight 2, so the degree of I_ℓ is exactly $\ell^4 - 1$.

From the computational point of view, a good choice for a generating set of I_ℓ is a Gröbner basis for a lexicographic order. Using the order $U_1 < U_0 < V_1 < V_0$, we can actually predict the shape of this Gröbner basis. Indeed, if D is an ℓ-torsion divisor, then its opposite $-D$ is also ℓ-torsion, so it has the same u-coordinates, and opposite v-coordinates. Furthermore, we make the genericity assumption that all the pairs $\{D, -D\}$ of ℓ-torsion divisors have different values for u_1. Then, the Gröbner basis for the ideal I_ℓ takes the form

$$I_\ell = \begin{cases} V_0 - V_1 S_0(U_1) \\ V_1^2 - S_1(U_1) \\ U_0 - R_0(U_1) \\ R_1(U_1), \end{cases}$$

where R_1 is a squarefree polynomial of degree $(\ell^4 - 1)/2$ and R_0, S_1, S_0 are polynomials of degree at most $(\ell^4 - 1)/2 - 1$. If such a Gröbner basis for I_ℓ is known, then it is not difficult to imitate Schoof's algorithm, by working in the quotient algebra $\mathbb{F}_p[U_1, U_0, V_1, V_0]/I_\ell$. Unfortunately, no easy computable recurrence formulae are known that relate Gröbner bases of ℓ-division ideals for different values of ℓ, just like for division polynomials of elliptic curves. Therefore we shall start with the approach of [11] using Cantor's division polynomials and show that we can derive efficiently a multiple of R_1.

3.1 Cantor's division polynomials

Let us fix a prime ℓ. Cantor's division polynomials [8] are polynomials in $\mathbb{F}_p[X]$, denoted by d_0, d_1, d_2, e_0, e_1, Δ, with the following property: for a divisor $P =$

$\langle x - x_P, y_P \rangle$ of weight 1, the multiplication of P by ℓ in $\mathbf{J}(\mathcal{C})$ is given by

$$[\ell]P = \left\langle x^2 + \frac{d_1(x_P)}{d_2(x_P)}x + \frac{d_0(x_P)}{d_2(x_P)},\ y_P\left(\frac{e_1(x_P)}{\Delta(x_P)}x + \frac{e_0(x_P)}{\Delta(x_P)}\right)\right\rangle.$$

These polynomials have respective degrees $2\ell^2-1$, $2\ell^2-2$, $2\ell^2-3$, $3\ell^2-2$, $3\ell^2-3$, $3\ell^2-2$ and are easily computed by means of recurrence formulae. Even if a naive method is used, the cost of their computation is by far negligible compared to the subsequent operations.

Now, let $D = \langle x^2+U_1x+U_0, V_1x+V_0\rangle$ be a generic divisor of weight 2, where U_1, U_0, V_1, V_0 are indeterminates, subject to the condition that $x^2 + U_1x + U_0$ divides $(V_1x + V_0)^2 - f$. The divisor D can be written as the sum of two weight 1 divisors $P_1 = \langle x - X_1, Y_1\rangle$ and $P_2 = \langle x - X_2, Y_2\rangle$, where $U_1 = -(X_1 + X_2)$, $U_0 = X_1X_2$, and where Y_1 and Y_2 satisfy $V_1X_1 + V_0 = Y_1$ and $V_1X_2 + V_0 = Y_2$. Since $D = P_1 + P_2$, then D is ℓ-torsion if and only if $[\ell]P_1 = -[\ell]P_2$.

Rewriting this equation using Cantor's division polynomials, we get four equations that must be satisfied for D to be ℓ-torsion. Some of these equations are multiples of $X_1 - X_2$: this is an artifact due to the splitting of D into divisors of weight 1 and if this is the case one should divide out this factor. Hence we obtain the following system:

$$\begin{cases} E_1(X_1, X_2) &= (d_1(X_1)d_2(X_2) - d_1(X_2)d_2(X_1))/(X_1 - X_2) = 0,\\ E_2(X_1, X_2) &= (d_0(X_1)d_2(X_2) - d_0(X_2)d_2(X_1))/(X_1 - X_2) = 0,\\ F_1(X_1, X_2, Y_1, Y_2) &= Y_1e_1(X_1)e_0(X_2) + Y_2e_1(X_2)e_0(X_1) = 0,\\ F_2(X_1, X_2, Y_1, Y_2) &= Y_1e_2(X_1)e_0(X_2) + Y_2e_2(X_2)e_0(X_1) = 0. \end{cases}$$

Consider now the finite-dimensional $\mathbb{F}_p$-algebra

$$B = \mathbb{F}_p[X_1, X_2, Y_1, Y_2]/(E_1, E_2, F_1, F_2, Y_1^2 - f(X_1), Y_2^2 - f(X_2)).$$

In a generic situation, the minimal polynomial of $-(X_1 + X_2)$ in B is then precisely the polynomial R_1 that appears in the Gröbner basis of I_ℓ (failures could occur, *e.g.*, if there exists an ℓ-torsion divisor $D = P_1 + P_2$, such that $[\ell]P_1$ is not of weight 2). We will see below that the whole Gröbner basis of I_ℓ is not necessary to the point-counting application we have in mind. Thus, we can start by working with the first two equations E_1, E_2, which involve X_1, X_2 only.

These polynomials were already considered in [11]. The strategy used in that paper consisted in computing the resultant of E_1, E_2 with respect to X_2 for a start, from which it was possible to deduce the coordinates of $[\ell]$-torsion divisors. This approach did not take into account the symmetry in (X_1, X_2); we now show how to work directly in Mumford's coordinates $U_1 = -(X_1 + X_2), U_0 = X_1X_2$, so as to compute resultants of lower degrees.

3.2 Resymmetrisation

The polynomials $E_1(X_1, X_2)$ and $E_2(X_1, X_2)$ are symmetric polynomials. It is well known that they can be expressed in terms of the two elementary symmetric

polynomials X_1X_2 and $X_1 + X_2$. The heart of Mumford's representation is the use of this expression, but this had been broken in order to apply Cantor's division polynomials. We call *resymmetrisation* the method that we present now to come back to a representation of bivariate polynomials in terms of the elementary symmetric polynomials. This is not as trivial as it seems, since the naive schoolbook method to symmetrize a polynomial yields a complexity jump in our case.

Let us consider the unique polynomials $\mathfrak{E}_1$ and $\mathfrak{E}_2$ in $\mathbb{F}_p[U_0, U_1]$ such that $\mathfrak{E}_1(X_1X_2, -X_1 - X_2) = E_1(X_1, X_2)$ and $\mathfrak{E}_2(X_1X_2, -X_1 - X_2) = E_2(X_1, X_2)$ and let $\overline{R_1} \in \mathbb{F}_p[U_1]$ be their resultant with respect to U_0; then R_1 divides $\overline{R_1}$.

We want to use the following evaluation/interpolation techniques to compute $\overline{R_1}$: evaluate the variable U_1 at sufficiently many scalars u_1, compute the resultants of $\mathfrak{E}_1(U_0, u_1)$ and $\mathfrak{E}_2(U_0, u_1)$, and interpolate the results. Unfortunately, computing with $\mathfrak{E}_1$ and $\mathfrak{E}_2$ themselves has prohibitive cost, as these polynomials have $O(\ell^4)$ monomials. However, their specific shape yields the following workaround.

Let h be a polynomial in $\mathbb{F}_p[X]$ and X_1 and X_2 be two indeterminates. Then the *divided differences* of h are the bivariate symmetric polynomials

$$A_0(h) = \Big(h(X_1) - h(X_2)\Big)/(X_1 - X_2) \text{ and } A_1(h) = \Big(X_1h(X_2) - X_2h(X_1)\Big)/(X_1 - X_2).$$

We let $\mathfrak{A}_0(h)$ and $\mathfrak{A}_1(h)$ be the unique polynomials in $\mathbb{F}_p[U_0, U_1]$ such that $\mathfrak{A}_0(h)(X_1X_2, -X_1 - X_2) = A_0(h)$ and $\mathfrak{A}_1(h)(X_1X_2, -X_1 - X_2) = A_1(h)$. Then a direct computation shows that

$$\begin{aligned}\mathfrak{E}_1 &= \mathfrak{A}_0(d_1)\ \mathfrak{A}_1(d_2) - \mathfrak{A}_0(d_2)\ \mathfrak{A}_1(d_1),\\ \mathfrak{E}_2 &= \mathfrak{A}_0(d_0)\ \mathfrak{A}_1(d_2) - \mathfrak{A}_0(d_2)\ \mathfrak{A}_1(d_0) \quad \text{in } \mathbb{F}_p[U_0, U_1].\end{aligned}$$

Given an arbitrary polynomial h in $\mathbb{F}_p[X]$ and $u_1 \in \mathbb{F}_p$, we show in the last paragraphs how to compute the polynomials $\mathfrak{A}_0(h)$ and $\mathfrak{A}_1(h)$ evaluated at $U_1 = u_1$ efficiently. Taking this operation for granted, we deduce Algorithm 2 for computing the resultant $\overline{R_1}$ of $\mathfrak{E}_1$ and $\mathfrak{E}_2$.

Algorithm 2 Computation of the resultant $\overline{R_1}$

1. For $\deg(\overline{R_1}) + 1$ different values of $u_1 \in \mathbb{F}_p$, do
   (a) Compute $\mathfrak{A}_0(d_0), \mathfrak{A}_1(d_0), \mathfrak{A}_0(d_1), \mathfrak{A}_1(d_1), \mathfrak{A}_0(d_2), \mathfrak{A}_1(d_2)$ evaluated at $U_1 = u_1$.
   (b) Deduce $\mathfrak{E}_1$ and $\mathfrak{E}_2$, evaluated at $U_1 = u_1$.
   (c) Compute $\overline{R_1}(u_1)$ as the resultant in U_0 of $\mathfrak{E}_1$ and $\mathfrak{E}_2$.
2. Interpolate $\overline{R_1}$ from the pairs $(u_1, \overline{R_1}(u_1))$.

The classical estimates for the degrees of resultants imply that the degree of $\overline{R_1}$ is $6\ell^4 - 17\ell^2 + 12$; thus to be able to perform the interpolation, it is necessary to take at least $6\ell^4 - 17\ell^2 + 13$ different values of u_1. In practice, it is recommended to take a few more values of u_1, in order to check the computation. Note that the resultant of E_1, E_2 has degree $8\ell^4 - 22\ell^2 + 15$.

We finish this subsection by detailing our solution to the problem raised above: given u_1 in $\mathbb{F}_p$ and h in $\mathbb{F}_p[X]$, how to compute the polynomials $\mathfrak{A}_0(h)$

and $\mathfrak{A}_1(h)$ evaluated at $U_1 = u_1$ efficiently? It is immediate to check the following identity:

$$h(X) = \mathfrak{A}_1(h)(U_0, u_1)X + \mathfrak{A}_0(h)(U_0, u_1) \mod (X^2 + u_1X + U_0).$$

Thus, the problem amounts to reduce h modulo $X^2 + u_1X + U_0$ in $\mathbb{F}_p[U_0][X]$. Our solution relies on the following primitive: If h is a polynomial of degree N in $\mathbb{F}_p[X]$ and a is a scalar in $\mathbb{F}_p$, then the coefficients of $h(X+a)$ can be deduced from the coefficients of $h(X)$ for one polynomial multiplication in degree N, see [2]. We call this primitive `var-shift`.

The main idea is now to rewrite the relation $X^2 + u_1X + U_0 = 0$ in the form $(X + u_1/2)^2 = u_1^2/4 - U_0$. Let $Y = X + u_1/2$, and k in $\mathbb{F}_p[X]$ such that $h(X) = k(Y)$. We group the coefficients of k according to the parity of their indices, forming the polynomials k_{odd} and k_{even} such that $k(Y) = k_{\text{even}}(Y^2) + Yk_{\text{odd}}(Y^2)$. Taking h modulo $X^2 + u_1X + U_0$, we have

$$h(X) \equiv k_{\text{even}}\left(\frac{u_1^2}{4} - U_0\right) + \left(X + \frac{u_1}{2}\right) k_{\text{odd}}\left(\frac{u_1^2}{4} - U_0\right).$$

Thus, computing $\mathfrak{A}_0(h)$ and $\mathfrak{A}_1(h)$ can be done by Algorithm 3 below.

Algorithm 3 Reduction of $h(X)$ modulo $X^2 + u_1X + U_0$ in $\mathbb{F}_p[U_0][X]$

1. Compute k from h using `var-shift`.
2. Decompose k in k_{odd} and k_{even}.
3. Compute $k_{\text{even}}\left(u_1^2/4 - U_0\right)$ and $k_{\text{odd}}\left(u_1^2/4 - U_0\right)$ using `var-shift`.
4. Recombine their coefficients to get $h(X) \bmod X^2 + u_1X + U_0$.

3.3 Parasites Prediction and Removal

In [11] it is shown that a factor of the resultant of E_1, E_2 can be predicted and used to speed-up the computation. This prediction is still possible in the context of the resymmetrisation, and the factor of $\overline{R_1}$ corresponding to such roots can be computed efficiently. The roots of this factor of $\overline{R_1}$ are called parasites: they are not the U_1-coordinates of an ℓ-torsion divisor, and actually appear as a by-product of our elimination scheme. Thus, they can be safely factored out.

If x_1 and x_2 in $\overline{\mathbb{F}_p}$ cancel d_2, then $E_1(x_1, x_2) = E_2(x_1, x_2) = 0$. The U_1 coordinates corresponding to these solutions can be written as $-(x_1 + x_2)$ where x_1 and x_2 are roots of d_2. Hence we obtain the following factor ρ of $\overline{R_1}$:

$$\rho(U_1) = \prod_{d_2(x_1)=0} \prod_{d_2(x_2)=0} (U_1 + x_1 + x_2).$$

The factor ρ is a *parasite*, as generically it does not lead to any ℓ-torsion divisor. Then ρ divides $\overline{R_1}$ but not R_1, so we lose nothing in eliminating it from $\overline{R_1}$. The polynomial ρ is computed using an algorithm of [4] dedicated to such questions.

Then Step 2. in Algorithm 2 is replaced by the interpolation of $\overline{R_1}/\rho$ from the pairs $(u_1, \overline{R_1}(u_1)/\rho(u_1))$.

The degree of ρ is $4\ell^4-12\ell^2+9$, so the degree of $\overline{R_1}/\rho$ is $2\ell^4-5\ell^2+3$, reducing by a factor of about 3 the number of values of u_1 that have to be considered in Algorithm 2. As an output, we now have at our disposal the polynomial $\mathfrak{R}_1 = \overline{R_1}/\rho$, which is a multiple of R_1. For comparison, the resultant computed in [11] has degree $4\ell^4 - 10\ell^2 + 6$, which is twice the degree of $\mathfrak{R}_1$.

3.4 Factorization and Reconstruction of a Torsion Element

Once the resultant $\mathfrak{R}_1$ has been computed, the task is not finished: indeed, what we want is the representation of an ℓ-torsion divisor, so that we can plug it into the equation of the Frobenius endomorphism. Here, there are two possible strategies:

1. Refine $\mathfrak{R}_1$ to get exactly R_1 and reconstruct from it the whole Gröbner basis of I_ℓ describing a generic ℓ-torsion divisor.
2. Look for small degree factors of $\mathfrak{R}_1$, check if they are indeed factors of R_1 and deduce the corresponding ℓ-torsion divisors.

By analogy with Schoof's algorithm for elliptic curves, one would think that the first choice is the most pertinent. However, refining $\mathfrak{R}_1$ into R_1 can be a costly task, and if there exist indeed small factors of $\mathfrak{R}_1$, then the second solution is faster. That is the reason why we chosen the second solution in our experiments described below. However, especially for $\ell = 17$ or 19, we could feel the limit of this choice. Therefore, for larger computations, we should probably switch to the first solution.

We now describe the second strategy with more details.

Let u_1 be a root of $\mathfrak{R}_1$ in an extension $\mathbb{F}_q$ of $\mathbb{F}_p$. We evaluate the polynomials E_1 and E_2 at $(X_1, -u_1 - X_1)$ in $\mathbb{F}_q[X_1]$, and obtain two univariate polynomials in $\mathbb{F}_q[X_1]$. Their GCD is (generically) a polynomial of degree 2 which might, or not, be the u-polynomial of an ℓ-torsion divisor. To settle the question, we take into account the last two equations F_1 and F_2, and check that our candidate u-polynomial is compatible with them. If not, we try again and select another root of $\mathfrak{R}_1$. Otherwise, we deduce the v-polynomial, and build an ℓ-torsion divisor defined over $\mathbb{F}_q$. It is then plugged into all possible candidates for the characteristic polynomial $\chi(T)$ mod ℓ to detect the right one.

We now concentrate on the problem of finding irreducible factors of $\mathfrak{R}_1$, using classical ingredients of polynomial factorization. It is interesting to find the factors of small degree first, as it reduces the subsequent computation. Thus, we start by detecting the linear factors, given by $\gcd(X^p - X, \mathfrak{R}_1(X))$. If this GCD is non-trivial, then the corresponding roots are separated and processed before maybe continuing the factorization. Then factors of degree d are detected for increasing d by computing $\gcd(X^{p^d} - X, \mathfrak{R}_1(X))$, and when we find a root, it is used to try to build an ℓ-torsion divisor that perhaps determines $\chi(T)$ mod ℓ.

This can be improved using the fact that the factorization pattern of R_1 is partly predictable. Indeed, due to the Galois structure induced by the group

law in $\mathbf{J}(\mathcal{C})$, some factorization patterns are forbidden. We can then proceed as follows: we first precompute the list of all possible patterns corresponding to ℓ and p, and we start looking for irreducible factors by increasing degree as before. At each step, the number of factors we find eliminates some patterns in the list. Then we look in the remaining patterns for the next smallest possible degree and try directly to catch factors of that degree. If there is a large gap between the current degree and the next one, the Baby-step/Giant-step strategy of [30] using modular compositions can yield a significant speed-up compared to the classical powering algorithm.

As another application of the factorization patterns, we mention the influence of the choice of p: if $p = 1 \bmod \ell$, then we can infer that the smallest irreducible factor of R_1 has degree at most $(\ell^2 + 1)/2$, compared to possibly $O(\ell^4)$ in the general case. We do not give details on the determination of the possible patterns for lack of space. The idea is similar to the one used in [12] for modular equations.

3.5 Complexity

We start by evaluating the cost in $\mathbb{F}_p$-operations of one iteration of Step 1 in Algorithm 2. Using Algorithm 3, the cost of computing $\mathfrak{A}_0(d_0)$, $\mathfrak{A}_1(d_0)$, $\mathfrak{A}_0(d_1)$, $\mathfrak{A}_1(d_1)$, $\mathfrak{A}_0(d_2)$, $\mathfrak{A}_1(d_2)$ is $O(\mathsf{M}(\ell^2))$, since the d_i have degree $O(\ell^2)$. Deducing $\mathfrak{E}_1$ and $\mathfrak{E}_2$ involves 4 more multiplications of polynomials of degree $O(\ell^2)$ at a cost of $O(\mathsf{M}(\ell^2))$. The resultant of $\mathfrak{E}_1$ and $\mathfrak{E}_2$ can then be computed using the HGCD algorithm at a cost of $O(\mathsf{M}(\ell^2)\log\ell)$.

Hence the resultant computation is dominating this step; this would not have been the case without the `var-shift` strategy. The loop in Step 1 must be repeated for $O(\ell^4)$ different values of u_1, so the cost of Step 1 is $O(\ell^4\mathsf{M}(\ell^2)\log\ell)$ operations in $\mathbb{F}_p$. Step 2 is a degree $O(\ell^4)$ polynomial interpolation, which can be done using $O(\mathsf{M}(\ell^4)\log\ell)$ operations in $\mathbb{F}_p$.

We now evaluate the influence of the parasite prediction on the complexity. The polynomial ρ is computed using the algorithm of [4] at a cost of $O(\mathsf{M}(\ell^4))$ operations. Then its evaluations at the $O(\ell^4)$ different values of u_1 can be deduced using $O(\mathsf{M}(\ell^4)\log\ell)$ operations in $\mathbb{F}_p$. Therefore, the cost of precomputing the effect of the parasite factor is negligible compared to the cost of computing $\overline{R_1}$.

Knowing the values of ρ on the different values of u_1 allows to interpolate a polynomial of degree 3 times less. This yields a speed-up by a factor at least 3 (and even more in practice, depending on the function M). Also the input of the factorization step is 3 times smaller, thus gaining a constant factor in that phase.

The factorization phase is less easy to analyze, since its complexity varies quite a lot depending on the degrees of the smallest irreducible factors. Denote by d the degree of the smallest factor of $\mathfrak{R}_1$ that allows to deduce $\chi(T) \bmod \ell$. By the powering algorithm, computing $\gcd(X^{p^d} - X, \mathfrak{R}_1(X))$ can be done using $O((d\log p + \log\ell)\mathsf{M}(\ell^4))$ operations in $\mathbb{F}_p$ and isolating one of the factors of degree d has similar expected complexity. From an irreducible factor of degree d, the

reconstruction of an ℓ-torsion divisor D defined over $\mathbb{F}_{p^d}$ requires to manipulate polynomials of degree $O(\ell^2)$ over $\mathbb{F}_{p^d}$, so it costs $O(\mathsf{M}(d\ell^2)\log\ell)$ operations in $\mathbb{F}_p$.

Finally, the detection of the invalid choices for $(s_1, s_2) \bmod \ell$ requires 4 applications of the Frobenius endomorphism to D and $O(\ell)$ group operations in $\mathbf{J}(\mathcal{C}/\mathbb{F}_{p^d})$, that is $O((\ell\log d + \log p)\mathsf{M}(d))$ operations in $\mathbb{F}_p$.

If d is small enough (say $d = O(\ell)$), this factoring strategy is satisfactory since its complexity is not worse than computing $\mathfrak{R}_1$, if $\log p$ is not too large. However, if d is $O(\ell^4)$, then the above complexity estimate of the factoring step is catastrophic. Using the known factorization patterns is useful in this context, even if the precise analysis is complicated. We expect that working with the whole ideal I_ℓ (thus avoiding the factorization) is more suited for a proper analysis; cleaning all details of that approach is out of the scope of this article.

4 Computation modulo small prime powers

Given a prime ℓ, from the knowledge of an ℓ-torsion divisor in $\mathbf{J}(\mathcal{C})$, one can deduce ℓ^2-torsion divisors by performing a division by ℓ in the Jacobian; iterating this process yields divisors of ℓ^3-torsion, ℓ^4-torsion, ... This can be used within Schoof's algorithm, so as to obtain modular information on the polynomial $\chi(T)$ modulo ℓ, ℓ^2, and so on. As appears below, there are many computational difficulties to overcome before this can be efficiently applied in practice. We mostly dedicated our efforts on the case $\ell = 2$, improving the techniques of [11], and spend much of this section describing this case. We thereafter briefly describe the case $\ell = 3$.

In the case $\ell = 2$, this lifting strategy was already used in [11]. It starts from the data of a 2-torsion divisor; then the iterative step is as follows. Suppose that a divisor D_k of 2^k-torsion is given; we denote by $\mathbb{F}_q$ the extension of the base field $\mathbb{F}_p$ over which D_k is defined. We make the assumption that D_k has weight 2, and write $D_k = \langle x^2 + u_1 x + u_0, v_1 x + v_0\rangle$.

There are exactly $2^4 = 16$ divisors D such that $[2]D = D_k$. Let us make the genericity assumption that all these divisors have weight 2, and introduce 4 indeterminates U_1, U_0, V_1, V_0 to denote the coordinates of D. Using doubling formulas coming from Cantor's addition algorithm, we obtain a system $\mathcal{F}_k$ that relates D and D_k:

$$\mathcal{F}_k \left| \begin{array}{ll} H_1(U_1,U_0,V_1,V_0) = u_1, & G_1(U_1,U_0,V_1,V_0) = 0,\\ H_2(U_1,U_0,V_1,V_0) = u_0, & G_2(U_1,U_0,V_1,V_0) = 0,\\ H_3(U_1,U_0,V_1,V_0) = v_1,\\ H_4(U_1,U_0,V_1,V_0) = v_0, \end{array}\right.$$

where H_1, H_2, H_3, H_4 are rational functions, and G_1, G_2 are polynomials which specify that $x^2+U_1x+U_0$ divides $(V_1x+V_0)^2-f$. Cleaning denominators, we are left with a polynomial system in U_1, U_0, V_1, V_0, with u_1, u, v_1, v_0 as parameters. We make the further genericity assumption that the ideal generated by $\mathcal{F}_k$ admits

a Gröbner basis of the form

$$\begin{cases} V_0 - L_0(U_1) \\ V_1 - L_1(U_1) \\ U_0 - M_0(U_1) \\ M_1(U_1), \end{cases}$$

where $M_1 \in \mathbb{F}_q[U_1]$ has degree 16 and L_0, L_1, M_0 have degree at most 15. Since D_k is 2^k-torsion, this provides a description of 16 divisors of 2^{k+1}-torsion.

The next step is to factorize the polynomial M_1 in $\mathbb{F}_q[U_1]$. Any factor of M_1 can be used to try and determine the characteristic polynomial χ mod 2^{k+1}, but some of them might give no information. Let r be one irreducible factor of lowest degree that allows the determination of χ mod 2^{k+1}, n its degree, and u_1 a root of r in $\mathbb{F}_{q^n}$. Then the divisor $D_{k+1} = \langle x^2 + u_1 x + M_0(u_1), L_1(u_1)x + L_0(u_1)\rangle$ is of 2^{k+1}-torsion. It can be used for the next loop of the algorithm.

From the computational point of view, the main tasks to perform at the kth step are the following: First, solve a zero-dimensional polynomial system of the form $[2]D = D_k$, then factorize a polynomial of degree 16. The following subsections detail our contributions on these questions. It should be clear that these computations are done with polynomials defined over an extension $\mathbb{F}_q$ of the base field $\mathbb{F}_p$, whose possibly high degree is the main cause of concern.

4.1 Performing a Division by 2

All the systems $\mathcal{F}_k$ that we consider are obtained in the same manner; as k grows, only their right-hand sides vary. The difficulty comes from the fact that the field of definition of u_1, u_0, v_1, v_0 is an extension of $\mathbb{F}_p$ of possibly high degree.

The solution, suggested in [11], is to solve the system $\mathcal{F}_k$ for generic values $\mathbf{u_1}, \mathbf{u_0}, \mathbf{v_1}, \mathbf{v_0}$. There are of course only two degrees of freedom, as $x^2 + \mathbf{u_1}x + \mathbf{u_0}$ must divide $(\mathbf{v_1}x + \mathbf{v_0})^2 - f$. Working over the base field $\mathbb{F}_p(\mathbf{u_1}, \mathbf{u_0})$, we are thus led to consider the system $\mathcal{F}_{\text{gen}}$ in the unknowns $\mathbf{v_1}, \mathbf{v_0}, U_1, U_0, V_1, V_0$

$$\mathcal{F}_{\text{gen}} \left| \begin{array}{ll} H_1(U_1, U_0, V_1, V_0) = \mathbf{u_1}, & G_1(U_1, U_0, V_1, V_0) = 0, \\ H_2(U_1, U_0, V_1, V_0) = \mathbf{u_0}, & G_2(U_1, U_0, V_1, V_0) = 0, \\ H_3(U_1, U_0, V_1, V_0) = \mathbf{v_1}, & G_1(\mathbf{u_1}, \mathbf{u_0}, \mathbf{v_1}, \mathbf{v_0}) = 0, \\ H_4(U_1, U_0, V_1, V_0) = \mathbf{v_0}, & G_2(\mathbf{u_1}, \mathbf{u_0}, \mathbf{v_1}, \mathbf{v_0}) = 0, \end{array} \right.$$

where the last two equations express that $x^2 + \mathbf{u_1}x + \mathbf{u_0}$ divides $(\mathbf{v_1}x + \mathbf{v_0})^2 - f$. Generically, the solutions of this system can be represented the following way:

$$\mathcal{T} \begin{cases} V_0 - L_0(U_1, \mathbf{v_1}), \\ V_1 - L_1(U_1, \mathbf{v_1}), \\ U_0 - M_0(U_1, \mathbf{v_1}), \\ M_1(U_1, \mathbf{v_1}), \\ \mathbf{v_0} - N_1(\mathbf{v_1}), \\ N_0(\mathbf{v_1}). \end{cases}$$

All these polynomials have coefficients in $\mathbb{F}_p(\mathbf{u_1}, \mathbf{u_0})$. The polynomial N_0 has degree 4, N_1 has degree less than 4, M_1 has degree 16 in U_1 and less than 4 in $\mathbf{v_1}$ and L_0, L_1, M_0 have degree less than 16 (resp. 4) in U_1 (resp. $\mathbf{v_1}$).

Systems like $\mathcal{F}_{\text{gen}}$ that involve free variables are difficult to handle. A direct application of a Gröbner basis algorithm over $\mathbb{F}_p(\mathbf{u_1}, \mathbf{u_0})$ fails by lack of memory, so we used the algorithm of [28], dedicated to such situations, to compute $\mathcal{T}$. Once $\mathcal{T}$ is known, it can be specialized on the coordinates of the divisor D_k, realizing its division by 2.

The solution presented in [11] followed the same approach, with a notable difference: instead of considering the representation $\mathcal{T}$, another representation was used, which involved polynomials of degree 64. Our approach reduces this degree to 16, and makes the subsequent computations much easier.

In terms of complexity, the polynomials defining the system $\mathcal{F}_{\text{gen}}$ have degree bounded independently from p; thus, computing $\mathcal{T}$ takes a bounded number of operations in $\mathbb{F}_p$. Next, at each division step, we must specialize $\mathbf{u_1}, \mathbf{u_0}, \mathbf{v_1}, \mathbf{v_0}$ on the coordinate of the divisor D_k in $\mathcal{T}$. If D_k is defined in a degree d extension of $\mathbb{F}_p$, then this substitution requires $O(\mathsf{M}(d))$ operations in $\mathbb{F}_p$.

4.2 Factorization using the Action of the 2-Torsion

After performing the division by 2, we are left with a description of the solution set V_k of the system $\mathcal{F}_k$ by means of the following representation:

$$M_1(U_1) = 0, \quad U_0 = M_0(U_1), \quad V_1 = L_1(U_1), \quad V_0 = L_0(U_1)$$

Now, we have to factorize the polynomial $M_1 \in \mathbb{F}_q[U_1]$. It has degree 16, which is moderate; the main issue is the degree of $\mathbb{F}_q$ over its prime field: in the computations presented below, $\mathbb{F}_q$ had degree up to 1280 on its prime field. We now show how to simplify this factorization, using the natural action of the 2-torsion group $\mathbf{J}(\mathcal{C})[2]$ on V_k, in the spirit of [14].

Let us see U_1 as a coordinate function on the set of weight 2 divisors (the choice of U_1 is arbitrary, but makes the computation easier). To any subgroup G of $\mathbf{J}(\mathcal{C})[2]$, we associate the averaging operator $\mathbf{S}_G : D \mapsto \sum_{g \in G} U_1(D+g)$, which is defined as soon as all divisors $D + g$ have weight 2. Now, G acts on V_k, and each orbit has cardinality $|G|$. The function $\mathbf{S}_G$ takes constant values on each orbit, so it takes at most $[\mathbf{J}(\mathcal{C})[2] : G]$ distinct values on V_k. By an additional genericity assumption, we may suppose that $\mathbf{S}_G$ takes precisely $[\mathbf{J}(\mathcal{C})[2] : G]$ distinct values on V_k.

To realize this algebraically, let us introduce the "divisor" $\mathbf{D}_0 = \langle x^2 + U_1 x + M_0, L_1 x + L_0 \rangle$, defined over $\mathbb{F}_q[U_1]/M_1$. Given any 2-torsion divisor g, we can apply the addition formulas to $\mathbf{D}_0$ and g, performing all operations in $\mathbb{F}_q[U_1]/M_1$ (the addition formulas require divisions, but if one of them fails it gives a proper factor of M_1). We obtain a "divisor" $\mathbf{D}_g = \langle x^2 + U_1^{(g)} x + U_0^{(g)}, V_1^{(g)} x + V_0^{(g)} \rangle$, where $U_1^{(g)}, U_0^{(g)}, V_1^{(g)}, V_0^{(g)}$ are in $\mathbb{F}_q[U_1]/M_1$; by construction, if D is any divisor in V_k, then the U_1-coordinate of $D + g$ is obtained by evaluating $U_1^{(g)}$ on the U_1-coordinate of D. Let thus $s_G = \sum_{g \in G} U_1^{(g)} \in \mathbb{F}_q[U_1]/M_1$. Then for any $D \in V_k$,

the value $\mathbf{S}_G(D)$ is obtained by evaluating s_G on the U_1-coordinate of D. From the above discussion on the function $\mathbf{S}_G$, we deduce that the minimal polynomial of s_G in $\mathbb{F}_q[U_1]/M_1$ has degree $[\mathbf{J}(\mathcal{C})[2] : G]$.

As an abstract group, $\mathbf{J}(\mathcal{C})[2]$ is isomorphic to $(\mathbb{Z}/2\mathbb{Z})^4$. Let us consider subgroups

$$G_1 \simeq (\mathbb{Z}/2\mathbb{Z}) \subset G_2 \simeq (\mathbb{Z}/2\mathbb{Z})^2 \subset G_3 \simeq (\mathbb{Z}/2\mathbb{Z})^3 \subset \mathbf{J}(\mathcal{C})[2] \simeq (\mathbb{Z}/2\mathbb{Z})^4.$$

Using the above construction, we associate to these subgroups the elements s_1, s_2, s_3 of $\mathbb{F}_q[U_1]/M_1$. Introducing their minimal polynomials, we deduce that the extension $\mathbb{F}_q \to \mathbb{F}_q[U_1]/M_1$ is isomorphic to the quotient of $\mathbb{F}_p[U_1, S_1, S_2, S_3]$ by some polynomials

$$\begin{cases} T_U(U_1, S_1, S_2, S_3) \\ T_1(S_1, S_2, S_3) \\ T_2(S_2, S_3) \\ T_3(S_3), \end{cases}$$

where all polynomials have degree 2 in their main variables, resp. S_3, S_2, S_1, U_1. Using this decomposition, we avoid the factorization of M_1: We start by factorizing T_3 over $\mathbb{F}_q$, and adjoin one of its roots to $\mathbb{F}_q$; then we factor T_2 over this new field, and so on. Thus, only the computation of T_3, T_2, T_1, T_U and four square root extractions are needed.

Suppose that $q = p^d$; then all polynomials T_3, T_2, T_1, T_U can be computed in $O(\mathsf{M}(d))$ operations in $\mathbb{F}_p$. For square-root extraction, we used a factorization algorithm quite similar to those of [33] and [17]. Using such algorithms, the expected complexity of extracting a square root in $\mathbb{F}_{p^d}$ is $O(C(d)\log(d) + \mathsf{M}(d)\log(p))$ operations in $\mathbb{F}_p$, where $C(d)$ denotes the cost of modular composition in degree d, so that $C(d) \in O(d^2 + \sqrt{d}\mathsf{M}(d))$, see [6]. One should note that this whole process only saves a constant factor over the factorization of M_1 from scratch; however, it was quite significant in practice.

In the worst case, after k lifting steps, the degree d might be of order $O(16^k)$. In this case, taking into account all previous estimates, the expected complexity to obtain a 2^k-torsion is expected to be in $O(kC(16^k) + \mathsf{M}(16^k)\log(p))$ base field operations. However, our experiments showed that with a surprising amount of uniformity, the degree of this extension was actually in $O(2^k)$, so the above complexity bound was by far overestimated.

4.3 Performing a Division by 3

Most of what was described above extends *mutatis mutandis* to arbitrary ℓ. Nevertheless, the computations become much more difficult: even for $\ell = 3$, we did not solve the system describing the division of a generic divisor by 3. Thus, we used the plain strategy to divide torsion divisors by 3, by means of successive Gröbner bases computations, over extensions of $\mathbb{F}_p$ of increasing degrees. As the tables below reveal, the time required for solving these polynomial systems makes this approach much more delicate than for 2-torsion. As a consequence, we did not implement the equivalent of our improved factorization process, and used a plain factorization strategy.

5 Implementation and Experiments

We implemented a whole point-counting algorithm including all the above-mentioned improvements and the MCT algorithm [21], first within the Magma computer algebra system [3]. Then, the critical parts of the computation modulo small primes and the MCT algorithm were implemented in C++ using the NTL library [29]. The communication between different parts of the program is done using files for small communications or named pipes in the case of a heavy interaction. For instance, the analysis of the factorization pattern of the resultant R_1 is implemented in a Magma program that sends elementary factoring tasks (like a modular composition) to a running NTL program.

To test our program we ran it on several randomly chosen curves defined over $\mathbb{F}_p$ with $p = 5 \times 10^{24} + 8503491$, with the hope to find some cryptographically secure Jacobians. An early abort strategy was used to eliminate curves $\mathcal{C}$ for which either the Jacobian order or the Jacobian order of the twisted curve was discovered to be non-prime. In particular, f must be irreducible to ensure the oddity of the group orders.

We have computed the characteristic polynomials of 32 randomly chosen curves, that yield 64 group orders, taking into account the twists. Due to the early abort strategy, these group orders are not divisible by any prime less than or equal to 19. Among them, 7 were found to be primes, meaning that the corresponding Jacobians are secure against all known attacks. One particular curve has the nice feature that both itself and its twist have a prime order Jacobian. The data for that curve can be found in the appendix.

Table 1 gives statistics for the runtimes of the different steps of the algorithm. They are given in seconds on a Pentium IV at 2.26 GHz having 1 GB of central memory. Due to the early abort strategy, the statistics for the factoring phase are made on less curves for larger ℓ, *e.g.*, 39 curves for $\ell = 5$, versus 21 curves for $\ell = 19$. More curves were computed on different computers and were not taken into account for the statistics.

The modular composition used for factorization is done using Brent and Kung's algorithm [6]. For $\ell = 17$ and $\ell = 19$, the precomputation (Baby steps) is not balanced with the Giant steps due to memory constraints. This explains why the runtimes for those values look so bad compared to other values.

As for the torsion lifting, 2-torsion was much easier to handle than 3-torsion, as we computed divisors of order $1024 = 2^{10}$, versus $27 = 3^3$ only. The curves we used were selected so that they have 8-torsion defined over $\mathbb{F}_{p^{10}}$ and 3-torsion defined over $\mathbb{F}_{p^4}$. Then in almost all cases, the 2^i-torsion divisors, $i \geq 3$, were defined in extensions of degrees $10, 20, 40, 80, \ldots$, and the 3^i-torsion divisors in extensions of degrees $4, 12, 36, \ldots$

After the modular computations, we know $\chi(T) \bmod 44696171520 = 2^{10} \cdot 3^3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19$; for comparison's sake, note that in [11], the modular computation went to $3843840 = 2^8 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13$. To conclude; we run the MCT algorithm. Due to memory requirements, we used a Xeon at 2.66 GHz with 2 GB of memory; this computation takes about 3 hours and 1.7 GB per curve.

Computations Modulo Small Primes							
ℓ	5	7	11	13	17	19	Theory
generic degree of $\mathfrak{R}_1$	1,128	4,560	28,680	56,280	165,600	258,840	$2\ell^4 - 5\ell^2 + 3$
generic degree of ρ	2,209	9,025	57,121	112,225	330,625	516,961	$4\ell^4 - 12\ell^2 + 9$
Time computing ρ	0.3	2	23	52	256	374	$O(\mathsf{M}(\ell^4)\log\ell)$
Time Step 1, Algo 1	6.5	63	1,504	5,072	34,869	69,162	$O(\ell^4\mathsf{M}(\ell^2)\log\ell)$
Time Step 2, Algo 1	0.3	2	17	39	182	275	$O(\mathsf{M}(\ell^4)\log\ell)$
Total time Algo 1	7.1	67	1,544	5,163	35,307	69,811	
Time $X^p \mod \mathfrak{R}_1$	3.3	15	77	280	2,251	2,294	$O(\mathsf{M}(\ell^4)\log p)$
Time Prec. Mod Comp	0.8	8	105	525	3,267	2,122	$O(\ell^2\mathsf{M}(\ell^4))$
Time Apply Mod Comp	1.1	11	225	976	20,768	51,710	$O(\ell^8 + \ell^2\mathsf{M}(\ell^4))$
Factoring Time (Min)	12.5	59	524	1,055	23,537	15,061	
Factoring Time (Max)	61	353	5,021	23,083	206,860	359,330	
Factoring Time (Avg)	42	193	2,700	9,415	117,785	145,734	

2- and 3- Torsion Lifting			
Torsion	Total Time (Min)	Total Time (Max)	Total Time (Avg)
27	10,901	11,317	11,511
1024	71,421	103,433	90,071

Lifting to 1024-torsion Details for a sample curve				
Generic Resolution: 5,104 sec				
Torsion	Degree	Specialization	Factor	Deducing χ
8	10	1	12	1
16	20	1	37	3
32	40	3	178	16
64	80	15	543	50
128	160	41	1,423	146
256	320	115	4,627	459
512	640	390	16,776	1,602
1024	1280	1301	58,408	6,590

Lifting to 27-torsion Details for a sample curve		
Torsion	9	27
Degree	12	36
Gröbner	745	3,811
Factor	914	5,917
Deducing χ	3	19

Table 1. Runtimes in seconds for the torsion computation on a 2.26 GHz Pentium IV.

Putting all this together, a complete point-counting for a random curve over $\mathbb{F}_p$ takes on average about 1 week.

For comparison, in the record-curve computation in [11] the Schoof-like part was used up to $\ell = 13$. Just the modulo 13 computation had taken 205 hours on a Pentium II at 450 MHz. A crude estimation gives a runtime of about 40 hours on the same computer as the one we used in this paper. This has to be compared with the 4 hour runtime that we obtained with our improvements and our new implementation.

Are the curves "random"? In our computer experiments, the "pure randomness" is biased in several places. Due to the cryptographical requirements, the

group order must be prime, so "random curve" should be understood as random among the curves with prime order Jacobians, but that is standard. Also a bias is introduced by our early abort strategy on both the curve and its twist.

A more important bias is in the choice of p. We choose a prime which is congruent to 1 modulo all the small primes ℓ for which we do the Schoof computation. This was meant to speed-up the factorization of the resultant $\mathfrak{R}_1$, as mentioned in Section 3.4. This dependency of the runtime of the algorithm in the form of p can be avoided by working in the formal algebra instead of factoring. In fact, in more recent versions of our software, we implemented this and the runtimes are slightly better for large ℓ. Hence, this bias could be removed.

The last bias that we introduced is the particular shape of the 8- and 3-torsion that we imposed. The goal was mostly to have the same kind of behavior for all the curves with respect to the division by 2 and by 3. Indeed, the division algorithms rely on Gröbner basis computations and are very hard to implement and to debug. The technical difficulty of handling our computation on many computers, with interactions between Magma and NTL interacting led us to add this simplification that made our code more reliable.

Our NTL implementation of the Schoof-like part has been made freely available [9]. The Magma implementation of the division algorithms is not stable enough to be exported in the present state.

6 Conclusion and Perspectives

In this paper, we have detailed algorithms used to compute the cardinalities of Jacobians defined over prime fields of order about 10^{24}. Most of our attention was aimed at improving the techniques for torsion computation introduced in [11].

We expect more improvements to be possible. For instance, for torsion index about 17 or 19, the factorization strategy of Subsection 3.4 becomes lengthy, and comparative tests with other strategies are necessary, possibly using the modular equations of [12]. Also, our techniques for lifting the 3-torsion are still quite crude, as we would like it to be as efficient as that of 2-torsion. We have designed a birthday paradox version of the MCT algorithm, to be described elsewhere [13], that loses a constant factor in runtime but is highly parallelisable and requires almost no memory. In future work, we also plan to use it on top of our torsion computation algorithms.

Acknowledgments. Many people have been of assistance when designing, implementing and running our algorithms. We thank Gérard Guillerm and Bogdan Tomchuk for letting classroom computers at École polytechnique at our disposal, Grégoire Lecerf for his fast evaluation and interpolation code, Allan Steel for releasing is HGCD implementation in Magma by our request, Nicolas M. Thiéry for sharing his insight on symmetric polynomials and Emmanuel Thomé for his help on handling large scale computations. The computations were performed on classroom computers at École polytechnique, on the machines of the MEDICIS center for computer algebra `http://www.medicis.polytechnique.fr/`, and on machines paid by ACI Cryptologie.

References

1. L. Adleman and M.-D. Huang. Counting points on curves and abelian varieties over finite fields. *J. Symbolic Comput.*, 32:171–189, 2001.
2. A. Aho, K. Steiglitz, and J. D. Ullman. Evaluating polynomials at fixed sets of points. *SIAM J. Comput.*, 4(4):533–539, 1975.
3. W. Bosma and J. Cannon. *Handbook of Magma functions*, 1997. `http://www.maths.usyd.edu.au:8000/u/magma/`.
4. A. Bostan, P. Flajolet, B. Salvy, and É. Schost. Fast computation with two algebraic numbers. Technical Report 4579, INRIA, 2002.
5. A. Bostan, P. Gaudry, and É. Schost. Linear recurrences with polynomial coefficients and computation of the Cartier-Manin operator on hyperelliptic curves. 2003. To appear in Proceedings Fq'7.
6. R. Brent and H. Kung. Fast algorithms for manipulating formal power series. *J. ACM*, 25:581–595, 1978.
7. D. G. Cantor. Computing in the Jacobian of an hyperelliptic curve. *Math. Comp.*, 48(177):95–101, 1987.
8. D. G. Cantor. On the analogue of the division polynomials for hyperelliptic curves. *J. Reine Angew. Math.*, 447:91–145, 1994.
9. P. Gaudry. *NTLJac2, Tools for genus 2 Jacobians in NTL.* `http://www.lix.polytechnique.fr/Labo/Pierrick.Gaudry/NTLJac2/`.
10. P. Gaudry and N. Gürel. Counting points in medium characteristic using Kedlaya's algorithm. To appear in *Experiment. Math.*
11. P. Gaudry and R. Harley. Counting points on hyperelliptic curves over finite fields. In W. Bosma, editor, *ANTS-IV*, volume 1838 of *Lecture Notes in Comput. Sci.*, pages 313–332. Springer–Verlag, 2000.
12. P. Gaudry and É. Schost. Modular equations for hyperelliptic curves. To appear in *Math. Comp.*
13. P. Gaudry and Éric Schost. A low-memory parallel version of Matsuo, Chao and Tsujii's algorithm. To appear in *ANTS VI*.
14. G. Hanrot and F. Morain. Solvability of radicals from an algorithmic point of view. In *ISSAC'01*, pages 175–182. ACM Press, 2001.
15. M. Hindry and J. Silverman. *Diophantine geometry. An introduction*, volume 201 of *Graduate Texts in Mathematics*. Springer–Verlag, 2000.
16. M.-D. Huang and D. Ierardi. Counting points on curves over finite fields. *J. Symbolic Comput.*, 25:1–21, 1998.
17. E. Kaltofen and V. Shoup. Fast polynomial factorization over high algebraic extensions of finite fields. In W. Kuchlin, editor, *ISSAC-97*, pages 184–188. ACM Press, 1997.
18. W. Kampkötter. *Explizite Gleichungen für Jacobische Varietäten hyperelliptischer Kurven.* PhD thesis, Universität Gesamthochschule Essen, August 1991.
19. T. Lange. Formulae for arithmetic on genus 2 hyperelliptic curves, 2003. Preprint.
20. R. Lercier and D. Lubicz. A quasi quadratic time algorithm for hyperelliptic curve point counting. Preprint.
21. K. Matsuo, J. Chao, and S. Tsujii. An improved baby step giant step algorithm for point counting of hyperelliptic curves over finite fields. In C. Fiecker and D. Kohel, editors, *ANTS-V*, volume 2369 of *Lecture Notes in Comput. Sci.*, pages 461–474. Springer-Verlag, 2002.
22. A. Menezes, Y.-H. Wu, and R. Zuccherato. An elementary introduction to hyperelliptic curves. In *Algebraic aspects of cryptography*, by N. Koblitz, pages 155–178, Springer-Verlag, 1997.

23. J.-F. Mestre. Utilisation de l'AGM pour le calcul de $E(\mathbb{F}_{2^n})$. Letter to Gaudry and Harley, December 2000.
24. J. S. Milne. Abelian varieties. In G. Cornell and J. H. Silverman, editors, *Arithmetic Geometry*, pages 103–150. Springer–Verlag, 1986.
25. J. Pelzl, T. Wollinger, J. Guajardo, and C. Paar. Hyperelliptic curve cryptosystems: Closing the performance gap to elliptic curves. Preprint, 2003.
26. J. Pila. Frobenius maps of abelian varieties and finding roots of unity in finite fields. *Math. Comp.*, 55(192):745–763, October 1990.
27. R. Schoof. Elliptic curves over finite fields and the computation of square roots mod p. *Math. Comp.*, 44:483–494, 1985.
28. É. Schost. Complexity results for triangular sets. *J. Symbolic Comput.*, 36:555–594, 2003.
29. V. Shoup. *NTL: A library for doing number theory.* `http://www.shoup.net/ntl/`.
30. V. Shoup. A new polynomial factorization algorithm and its implementation. *J. Symbolic Comput.*, 20:363–397, 1995.
31. F. Vercauteren. Computing Zeta functions of hyperelliptic curves over finite fields of characteristic 2. In M. Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Comput. Sci.*, pages 369–384. Springer-Verlag, 2002.
32. J. von zur Gathen and J. Gerhard. *Modern computer algebra*. Cambridge University Press, 1999.
33. J. von zur Gathen and V. Shoup. Computing Frobenius maps and factoring polynomials. *Comput. Complexity*, 2:187–224, 1992.
34. A. Weng. Constructing hyperelliptic curves of genus 2 suitable for cryptography. *Math. Comp.*, 72:435–458, 2003.

Appendix: A cryptographically secure curve

Let $\mathcal{C}$ be defined by $y^2 = f(x)$ over $\mathbb{F}_p$ with $p = 5 \times 10^{24} + 8503491$, and

$$f(x) = x^5 + 268281082283935564490073 6x^3 + 226591355295993102902116x^2 + 254767471595292971789991 8x + 479730995970848967305935 0.$$

Then its characteristic polynomial is $\chi(T) = T^4 - s_1T^3 + s_2T^2 - ps_1T + p^2$, where

$$s_1 = 1173929286783 \quad \text{and} \quad s_2 = 4402219446392186881834853.$$

Thus the cardinality of its Jacobian is

$$N_{\mathbf{J}} = \chi(1) = 24999999999994130438600999402209463966197516075699,$$

which is a 164-bit prime number. Furthermore the quadratic twist of $\mathcal{C}$ has a Jacobian with group order

$$N_{\tilde{\mathbf{J}}} = \chi(-1) = 25000000000005869731468829402229428962794965968171,$$

which is also a prime number.