Genus 2 point counting over prime fields

Pierrick Gaudry

LORIA, CNRS / INRIA / Nancy Université, Nancy, France

Éric Schost

Computer Science Department, The University of Western Ontario, London ON

Abstract

For counting points of jacobians of genus 2 curves over a large prime field, the best known approach is essentially an extension of Schoof's genus 1 algorithm. We propose various practical improvements to this method and illustrate them with a large scale computation: we counted hundreds of curves, until one was found that is suitable for cryptographic use, with a state-of-the-art security level of approximately 2^{128} and desirable speed properties. This curve and its quadratic twist have a Jacobian group whose order is 16 times a prime.

Key words: Point couting, hyperelliptic curves, Schoof-Pila algorithm.

1. Introduction, previous work

For a given level of security, genus 2 curves can now provide cryptosystems that are competitive with their elliptic analogues in terms of speed (see for instance Cohen and Frey (2005) for a general introduction to elliptic and hyperelliptic curve cryptography). However, in contrast to the elliptic case, it remains difficult to construct secure genus 2 cryptosystems. To wit, we have the following requirements:

- the base field should be large enough: a field cardinality of 2^{80} is believed to provide barely adequate security, and 2^{128} is considered safe;
- the curve (and possibly its twist) should have a Jacobian of prime, or almost prime group order (a few small factors may be acceptable).

We will call a curve that satisfies these constraints a (twist-) secure curve. Two approaches coexist to obtain such curves: point-counting, and construction using the complex multiplication method (see for instance Streng's thesis (Streng, 2010)). In this paper,

Email addresses: pierrick.gaudry@loria.fr (Pierrick Gaudry), eschost@uwo.ca (Éric Schost).

motivated by efficiency considerations (described in detail in the last section), we choose the former.

When the base field has a small characteristic, very efficient algorithms have been designed and implemented, based on a p-adic lifting of the curve, after the work of Satoh (2000) and Kedlaya (2001), or deformation techniques following Lauder (2004) – we refer to the survey articles (Chambert-Loir, 2008; Gaudry, 2006) as well as Cohen and Frey (2005, Ch. 17) for further details and references. Unfortunately, the complexity of the p-adic algorithms is exponential in $\log(p)$ (it has been lowered to $\sqrt{p}$ in the work of Harvey (2007)), so we cannot apply them for a large prime field.

In that case, the approach is essentially an extension of Schoof's genus 1 algorithm, that appeared first in work of Pila (1990) (for the very general case of an abelian variety), followed by Huang and Ierardi (1998) and Adleman and Huang (2001); all these algorithms have a runtime polynomial in $\log(p)$. The special case of genus 2 is discussed by Gaudry and Harley (2000), the authors (Gaudry and Schost, 2004a), and Pitcher (2009).

We mention a third approach to produce good curves, due to Sutherland (2009), that is not really point counting: using generic group algorithms, it is possible to produce in subexponential time a curve with a Jacobian of known group order. Unfortunately, it seems that there is no way to turn it into a polynomial time algorithm. Furthermore, Sutherland's technique cannot output a twist-secure curve.

Schoof's algorithm. To find the cardinality of the Jacobian of a curve $\mathcal{C}$, the key idea of Schoof's algorithm is to compute the characteristic polynomial $\chi \in \mathbb{Z}[T]$ of the Frobenius endomorphism modulo several small prime numbers ℓ, and to reconstruct χ by the Chinese Remainder Theorem, using Weil's bounds on its coefficients.

For a given ℓ, the ℓ-torsion subgroup of the Jacobian of $\mathcal{C}$ is a finite group isomorphic to $(\mathbb{Z}/\ell\mathbb{Z})^4$, and the action of the Frobenius endomorphism on it is $\mathbb{Z}/\ell\mathbb{Z}$-linear. Computing explicitly this subgroup and the action of the Frobenius on it therefore provides us with the characteristic polynomial χ modulo ℓ. The most difficult part of Schoof's algorithm in genus 2 is the explicit computation of the torsion subgroup.

It is also possible to combine information modulo prime powers ℓ^k, for very small ℓ, say up to 7. This is obtained by constructing elements of the ℓ^k-torsion subgroup, on which we can test the action of the Frobenius. Again, the costly part is to construct these torsion elements.

If we run out of feasible primes, or powers of small primes, and we do not have enough modular information to reconstruct χ unambiguously, we deduce χ using a matching algorithm such as the ones of Matsuo et al. (2002) or Gaudry and Schost (2004b) (this will be the case for our experiments).

Under a few non-degeneracy assumptions, the state-of-the-art approach to compute the ℓ-torsion takes $O(\ell^6)$ operations in $\mathbb{F}_p$ (ignoring logarithmic factors); taking all required ℓ's into account results in $O(\log(p)^7)$ operations in $\mathbb{F}_p$. Remark that from a purely theoretical point of view, using prime powers only changes the constant factor in the big-O; however it makes an important difference in practice.

In the case of genus 1, the improvements by Elkies and Atkin provide a way to deal with only part of the ℓ-torsion. This method makes heavy use of modular equations and explicit isogenies. Despite several works attempting to extend these tools (Gaudry and Schost, 2005; Bröker and Lauter, 2009; Faugère et al., 2009; Lubicz and Robert, 2010), in genus 2 there is still no known algorithm for computing part of the ℓ-torsion faster than the whole torsion. Therefore, in this work we do not follow this direction.

Our contribution. Our purpose in this article is to give a detailed presentation of the algorithm sketched above and of our implementation, dedicated to genus 2 curves over the prime field $\mathbb{F}_p$.

We present several improvements upon previous work (Gaudry and Harley, 2000; Gaudry and Schost, 2004a), that mainly concern the construction of torsion elements, either ℓ-torsion, or ℓ^k-torsion when ℓ is tiny. Our contributions do not allow us to reduce the exponent 7 of the complexity of the algorithm; however, we put a significant effort into saving all possible constant factors. This is necessary to reach cryptographic size.

For a base field of size about 128 bits, a natural choice is to work in $\mathbb{F}_p$, with $p = 2^{127} - 1$. Over this base field, our implementation allows us to compute the cardinality of a curve in about 1000 CPU hours; as far as we can tell, this is the first time that one achieves genus 2 point-counting over such a field: previous landmarks for prime fields were $p \approx 2^{61}$ (Gaudry and Harley, 2000) and $p \approx 2^{82}$ (Gaudry and Schost, 2004a). A large-scale deployment of our implementation, coupled with early abort strategies, enabled us to find the first twist-secure curve, that also possesses desirable speed properties; the computation took more than 1,000,000 CPU hours.

To our knowledge, no other published work gives a precise description of this kind of implementation; our goal for this paper is to fill this gap, and provide all necessary details. There is a moderate price to pay: some claims below (such as the shape of some Gröbner bases, the nature of the parasite factors in our equations, etc) are stated without proof, but are backed up by the fact that they held in our experiments. This can arguably be a sufficient justification from the practical point of view; in theory, in most cases, genericity arguments could be used to prove that our claims hold for a generic curve.

Notation. In all that follows, $\mathcal{C}$ is a genus 2 curve with Weierstraß equation $Y^2 = f(X)$, where f is monic, of degree 5, over a prime field $\mathbb{F}_p$, with $p > 5$. Its Jacobian variety is denoted by $\mathcal{J} = \mathrm{Jac}(\mathcal{C})$; it is the degree zero divisor class group of $\mathcal{C}$. The point at infinity on $\mathcal{C}$ is written ∞.

A non-zero element D of $\mathcal{J}$ can be uniquely written $D = \langle U(X), V(X)\rangle$, with U monic, and with either $\deg(U) = 2$ and $\deg(V) \leq 1$, or $\deg(U) = 1$ and $\deg(V) \leq 0$.

- In the former case (which is the generic case), we say that D has *weight 2*. Then, D can be written as $P_1 + P_2 - [2]\infty$, where $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ are the two points on $\mathcal{C}$ such that $U(x_i) = 0$ and $V(x_i) = y_i$, for $i = 1, 2$. The case $P_1 = P_2$ can be dealt with by properly handling multiplicities.
- In the latter case, where $\deg(U) = 1$, we say that D has *weight 1*; it is of the form $D = P - \infty$, where $P = (x, y)$ is the point on $\mathcal{C}$ such that $U(x) = 0$ and $V(x) = y$.

In both cases, the conditions given above amount to $(V^2 - f) = 0 \bmod U$. This representation is called the *Mumford representation*, with the two polynomials in it respectively called the U-polynomial and V-polynomial of D; the field of definition of D is the field generated by the coefficients of U and V.

An algorithm due to Cantor allows one to compute the group law with this representation of elements of $\mathcal{J}$. We refer to Cohen and Frey (2005) for background on this explicit way of computing with Jacobians.

The pth power Frobenius automorphism $\pi : \overline{\mathbb{F}_p} \to \overline{\mathbb{F}_p}$ is extended to the Jacobian, and is still denoted by π. In the ring of endomorphisms of $\mathcal{J}$, it admits a characteristic polynomial of the form

$$\chi(T) = T^4 - s_1 T^3 + s_2 T^2 - p s_1 T + p^2 \in \mathbb{Z}[T], \tag{1}$$

where s_1 and s_2 are integers that satisfy

$$|s_1| \leq 4\sqrt{p} \quad \text{and} \quad |s_2| \leq 6p.$$

Since $|\mathcal{J}(\mathbb{F}_p)| = \chi(1)$, we will focus on computing χ, that is, s_1 and s_2. Note that the bound on s_2 can be refined to $2|s_1|\sqrt{p} - 2p \leq s_2 \leq \frac{s_1^2}{4} + 2p$, which yields a small practical speed-up (Lenstra, Jr. et al., 2002).

Organization of the paper. We give in Section 2 a review of algorithms for univariate and bivariate polynomials: they are the key ingredients for what follows. The core of this paper is in Sections 3 and 4, which explain in detail how to compute ℓ-torsion and ℓ^k-torsion divisors, and how to deduce χ mod ℓ from this data. Finally, Section 5 presents the computation that led to the discovery of a twist-secure curve in cryptographic size.

Acknowledgments. This work was made possible by the facilities of the Shared Hierarchical Academic Research Computing Network (SHARCNET: `www.sharcnet.ca`) and Compute / Calcul Canada, by means of a Dedicated Resources award. We also acknowledge financial support from INRIA, NSERC and the Canada Research Chairs program.

We wish to thank Daniel Bernstein and Nicole Pitcher for several suggestions, Baolai Ge for his support with the SHARCNET clusters and the reviewers for their constructive comments.

2. Algebraic algorithms

In this section, we present some (mostly classical) results about polynomial arithmetic: in the first subsection, we review known material for problems such as multiplication, composition, etc; in the second subsection, we discuss in more detail the operations used to handle base field extensions.

Especially in the second subsection, some of our algorithmic choices are dictated by practical considerations: we present the solutions that were found to be the most efficient using the library NTL (Shoup, 1995, 1996–2010), which forms the basis of our implementation. For the same reason, costs are given in terms of operations in $\mathbb{F}_p$: this analysis reflects rather closely the behavior of NTL for the problem sizes we consider.

Note that another point of view is possible, using the Kronecker-Schönhage substitution to reduce multiplication in $\mathbb{F}_p[X]$ to integer multiplication, foregoing polynomial arithmetic. Using this idea would allow one to save a factor $\log\log(p)$ in the overall bit complexity of the point counting algorithm, as pointed out by Pitcher (2009). However, our approach allowed us to rely on the large number of (well optimized) preexisting functions present in NTL.

2.1. Basic algorithms

Multiplication. We let M be such that polynomials of degree less than n in $\mathbb{F}_p[X]$ can be multiplied in $\mathsf{M}(n)$ operations in $\mathbb{F}_p$; we also add the super-linearity constraints of Gathen and Gerhard (1999, Ch. 8). Using Fast Fourier Transform, one can take $\mathsf{M}(n)$ in $O(n\log(n))$, provided $\mathbb{F}_p$ contains a primitive nth root of unity, and $O(n\log(n)\log\log(n))$ in general.

Then, for P of degree n in $\mathbb{F}_p[X]$, multiplication in $\mathbb{F}_p[X]/\langle P\rangle$ takes time $O(\mathsf{M}(n))$; inversion, if possible, takes time $O(\mathsf{M}(n)\log(n))$. The finite field $\mathbb{F}_{p^n}$ will be described as $\mathbb{F}_{p^n} = \mathbb{F}_p[X]/\langle P\rangle$, where $P \in \mathbb{F}_p[X]$ is an irreducible polynomial of degree n. This way, additions in $\mathbb{F}_{p^n}$ take time $O(n)$, multiplications $O(\mathsf{M}(n))$ and inversions $O(\mathsf{M}(n)\log(n))$.

Modular composition. Take P squarefree of degree n in $\mathbb{F}_p[X]$, not necessarily irreducible, and write $\mathbb{A} = \mathbb{F}_p[X]/\langle P\rangle$. On input $Z \in \mathbb{A}$ and $F \in \mathbb{F}_p[X]$ of degree e, *modular composition* is the problem of computing $F(Z) \in \mathbb{A}$.

In what follows, we will let $\mathsf{C}(e, n)$ be such that this operation can be done in $\mathsf{C}(e, n)$ operations in $\mathbb{F}_p$, and write $\mathsf{C}(n) = \mathsf{C}(n, n)$. Using the algorithm of Brent and Kung (1978), one can take $\mathsf{C}(e, n) = O(\mathsf{M}(n)e^{1/2} + ne^{(\omega-1)/2})$, where ω is such that matrices of size n can be multiplied in $O(n^\omega)$ operations; this algorithm has a memory requirement of $O(ne^{1/2})$ elements of $\mathbb{F}_p$, which can become a bottleneck.

The function $\mathsf{C}(n)$ can (in theory) be taken subquadratic, using fast matrix multiplication; however, the NTL implementation we use has $\omega = 3$, whence a quadratic behavior. In view of this estimate, we will make the assumption that $\mathsf{M}(n)\log(n)$ is $O(\mathsf{C}(n))$. Note that the algorithm of Kedlaya and Umans (2008) has a bit complexity quasi-linear in $\max(n, e)$. In theory, using this algorithm would reduce to $1 + o(1)$ the exponent of all algorithms involving factorization and root-finding that appear in Section 4; this would however not affect the overall running time, which is asymptotically dominated by that of the large ℓ case. From the practical point of view, we did not use the Kedlaya-Umans algorithm in our experiments, since we do not know of a competitive implementation of it.

Modular composition (and a "dual" problem, called power projection) are used in many further algorithms, as illustrated in the two results below. Both are standard, and are easily deduced from Shoup (1994) and Rouillier (1999).

- The *minimal polynomial* of an element $Z \in \mathbb{A}$ (that is, the minimal polynomial of the multiplication-by-Z map) can be computed in time $O(\mathsf{C}(n))$, provided $p > n$ (by first computing the characteristic polynomial of Z and taking its squarefree part).
- Even though $\mathbb{A}$ may not be a field, we call $Z \in \mathbb{A}$ a *primitive element* if its powers form an $\mathbb{F}_p$-basis of $\mathbb{A}$, that is, if its minimal polynomial has maximal degree n. In this case, given $Z' \in \mathbb{A}$, one can compute $S \in \mathbb{F}_p[X]$ such that $Z' = S(Z)$ using $O(\mathsf{C}(n))$ operations in $\mathbb{F}_p$.

Extensions to bivariate computations. Similar results hold for bivariate computations: for P as above, and Q in $\mathbb{F}_p[X, Y]$, of degree less than n in X and monic of degree m in Y, multiplication in $\mathbb{B} = \mathbb{F}_p[X, Y]/\langle P, Q\rangle$ can be done in time $O(\mathsf{M}(nm))$, see Gathen and Shoup (1992).

The notion of modular composition carries over: given F of degree e in $\mathbb{F}_p[X]$ and Z in $\mathbb{B}$, it consists in computing $F(Z) \in \mathbb{B}$. We will make the assumption that the function C is such that this can be done in time $\mathsf{C}(e, mn)$; this is indeed the case when using a bivariate version of the Brent and Kung algorithm, with $\mathsf{C}(e, mn) = O(\mathsf{M}(nm)e^{1/2} + mne^{(\omega-1)/2})$, see C. Pascal and Schost (2006). In this case, the memory requirement is $O(mne^{1/2})$ elements of $\mathbb{F}_p$.

The notions of minimal polynomial and primitive element are defined as before. When the ideal $\langle P, Q\rangle$ is radical, and $p > nm$, using techniques that extend the univariate ones, it is then possible to compute the minimal polynomial of an element $Z \in \mathbb{B}$, and, if Z is a primitive element, to express any $Z' \in \mathbb{B}$ as a polynomial in Z, in time $O(\mathsf{C}(mn))$.

However, one should note that using NTL, bivariate operations are slower than univariate ones by a rather large constant factor (e.g., with $p = 2^{127} - 1$, bivariate modular multiplication is 5 to 6 times as slow as univariate modular multiplication for similar input size). This remark will dictate some of the choices made in the next subsection.

Miscellaneous operations. Evaluation and interpolation of polynomials in degree n can be done in $O(\mathsf{M}(n)\log(n))$ operations, using subproduct tree techniques (Gathen and Gerhard, 1999). If one can choose the evaluation points, it is possible to do better: using points in geometric progression, one can reduce both costs to $2\mathsf{M}(n) + O(n)$, see Bostan and Schost (2005). Besides, the memory usage is then linear in n (assuming polynomial multiplication is done in linear space); this can be achieved as well using subproduct tree techniques, but not in a straightforward manner (Gathen and Shoup, 1992).

The next operation is less well-known. The *sum-root polynomial* of two polynomials F and G is the polynomial whose roots are the sums of one root of F and one root of G:

$$\mathrm{SR}(F,G) = \prod_{F(x_1)=0} \prod_{G(x_2)=0} (X - (x_1 + x_2)).$$

In the case where $F = G$, we can isolate the contribution coming from $x_1 = x_2$ and define the reduced sum-root polynomial $\mathrm{sr}(F)$ by the relation:

$$\mathrm{SR}(F,F) = F(X/2)\ \mathrm{sr}(F)^2.$$

One can compute $\mathrm{SR}(F,G)$ in time $O(\mathsf{M}(nm))$, with $\deg(F) = n$ and $\deg(G) = m$, provided $p > nm$, see Bostan et al. (2006).

Finally, we discuss how to shift the variable in a polynomial. If H is in $\mathbb{F}_p[X]$ of degree n and a is in $\mathbb{F}_p$, then the coefficients of $H(X + a)$ can be deduced from the coefficients of H in time $\mathsf{M}(n) + O(n)$, assuming $p > n$, using the algorithm of Aho et al. (1975).

Solving bivariate systems. Let now A and B be two polynomials in $\mathbb{F}_p[X,Y]$, and let $d_A = \deg(A,Y)$ and $d_B = \deg(B,Y)$; we assume that $d_A \geq d_B$ (if not, exchange the roles of these polynomials). To such a system, we associate the *resultant* and the *subresultant of degree 1* of A and B with respect to Y, written $R = \mathrm{res}(A,B,Y)$ and $S = \mathrm{sres}(A,B,Y)$. Following Reischert (1997, Def. 3.1), for $i \in \mathbb{Z}$, let a_i (resp. b_i) be the coefficient of Y^i in A (resp. B), with $a_i = 0$ for $i < 0$ or $i > d_A$, and similarly for b_i. Then, R and S are the determinants of the matrices M_R and M_S given by

$$M_R = \begin{vmatrix} a_{d_A} & & & b_{d_B} & & \\ a_{d_A-1} & \ddots & & b_{d_B-1} & \ddots & \\ \vdots & \ddots & a_{d_A} & \vdots & \ddots & b_{d_B} \\ \vdots & & a_{d_A-1} & \vdots & & b_{d_B-1} \\ \vdots & & \vdots & \vdots & & \vdots \\ a_{2-d_B} & & a_2 & b_{2-d_A} & & b_2 \\ Y^{d_B-1}A & \cdots & Y^0 A & Y^{d_A-1}B & \cdots & Y^0 B \end{vmatrix}$$

and

$$M_S = \begin{vmatrix} a_{d_A} & & & b_{d_B} & & \\ a_{d_A-1} & \ddots & & b_{d_B-1} & \ddots & \\ \vdots & \ddots & a_{d_A} & \vdots & \ddots & b_{d_B} \\ \vdots & & a_{d_A-1} & \vdots & & b_{d_B-1} \\ \vdots & & \vdots & \vdots & & \vdots \\ a_{4-d_B} & & a_2 & b_{4-d_A} & & b_2 \\ Y^{d_B-2}A & \cdots & Y^0A & Y^{d_A-2}B & \cdots & Y^0B \end{vmatrix}.$$

Elementary row combinations show that R is in $\mathbb{F}_p[X]$, and S in $\mathbb{F}_p[X,Y]$, of the form $S = S_0(X) + S_1(X)Y$.

The polynomials R and S will be our basic tools to solve the system $A = B = 0$. Since they are in the ideal $\langle A, B\rangle$, any solution of $A = B = 0$ is a solution of $R = S = 0$. Conversely, the specialization property of (sub)resultants implies that any solution (x, y) of $R = S = 0$, with in addition $S_1(x) \neq 0$, is a solution of $A = B = 0$. When we deal with bivariate systems, we will be interested only in such solutions. Roughly speaking, these are the points $(x, y) \in \overline{\mathbb{F}_p}^2$ where x does not cancel both leading coefficients of A and B in Y, and such that there is no other solution of the form (x, y'). For a "random" system, we obtain all solutions this way.

To compute R and S, we will use evaluation and interpolation techniques: for sufficiently many values x_i, we compute the resultant and the subresultant of degree 1 of $A(x_i, Y)$ and $B(x_i, Y)$, provided that x_i does not cancel the leading coefficient of A or B in Y. If both A and B have total degree n, R and S have degrees at most n^2. The required (sub)resultants of $A(x_i, Y)$ and $B(x_i, Y)$ can be computed in $O(n^2\mathsf{M}(n)\log(n))$ operations, by an extension of the half-GCD algorithm, see Reischert (1997). Using points in a geometric progression, the evaluations can be done in $O(n\mathsf{M}(n^2))$ operations, and the interpolation in $O(\mathsf{M}(n^2))$ operations.

2.2. Managing field extensions

The algorithms for computing ℓ^k-torsion divisors will require us to extend the current base field, say $\mathbb{F}_{p^n}$, by adjoining to it a root γ of a polynomial $A \in \mathbb{F}_{p^n}[Y]$. This problem is especially important in the case of 2^k-torsion, and to a lesser extent, 3^k, 5^k and 7^k-torsion (these algorithms have several other potential bottlenecks). In most cases, $d = \deg(A, Y)$ is small; as a consequence, improvements for the case of large d are not discussed here.

Starting from $\mathbb{F}_{p^n}$ given as $\mathbb{F}_p[X]/\langle P\rangle$, we will have to find a univariate polynomial defining $\mathbb{F}_{p^m} = \mathbb{F}_{p^n}(\gamma)$ over $\mathbb{F}_p$, and to be able to apply the embedding $\mathbb{F}_{p^n} \to \mathbb{F}_{p^m}$. We present here a solution which requires us to factor only univariate polynomials over $\mathbb{F}_p$ (this is interesting for us, as we have mentioned that NTL does better at arithmetic in $\mathbb{F}_p[Y]$ than in $\mathbb{F}_{p^n}[Y]$). Our algorithm runs in expected time $O(\mathsf{C}(n)\log(n)+\mathsf{M}(n)\log(p))$, for fixed d.

These ideas are not new, as they already appear in the algorithm of Trager (1976) (except for small refinements in a special case, which we discuss below). The only difference is in the cost analysis: Trager's presentation relied on resultant techniques; following

Gathen and Shoup (1992), Shoup (1994) and Kaltofen and Shoup (1997), we use modular composition, which yields a faster algorithm (this idea appears also in Couveignes and Lercier (2009), where it is used in conjunction with Kedlaya and Umans' modular composition algorithm). Proofs not given below can be found in Trager (1976).

Overview of the algorithm. As input, we are given an irreducible polynomial $P \in \mathbb{F}_p[X]$ of degree n, as well as a monic squarefree polynomial $A \in \mathbb{A}[Y]$, where we write $\mathbb{A} = \mathbb{F}_p[X]/\langle P\rangle \simeq \mathbb{F}_{p^n}$. We want to find an extension $\mathbb{F}_{p^n} \to \mathbb{F}_{p^m}$, such that $\mathbb{F}_{p^m}$ contains a root of A. We write $d = \deg(A, Y)$, and we assume $p > nd$.

Let $I \subset \mathbb{F}_p[X, Y]$ be the ideal $\langle P, A\rangle$ and let $\mathbb{B} = \mathbb{F}_p[X, Y]/I$; remark that $\mathbb{B}$ has dimension nd over $\mathbb{F}_p$. Take Z of the form $Z = Y + rX$, for some $r \in \mathbb{F}_p$ and compute its minimal polynomial $Q \in \mathbb{F}_p[X]$. We want Z to be a primitive element. This will be the case for most choices of r: this condition is equivalent to Q having degree nd, and there are at most n^2d^2 choices of r for which this fails.

By the results recalled in Subsection 2.1, the cost of computing Q is $O(\mathsf{C}(nd))$, which is $O(\mathsf{C}(n))$ when d is fixed. If $\deg(Q) = nd$, we can compute S and T in $\mathbb{F}_p[X]$ such that $X = T(Z) \bmod Q$ and $Y = S(Z) \bmod Q$ for the same cost; if not, we start over with another choice of Z.

Assuming that $\deg(Q) = nd$, if Q is not irreducible, we furthermore replace Q, S, T by Q_1, S_1, T_1, where Q_1 is an irreducible factor of Q, $S_1 = S \bmod Q_1$ and $T_1 = T \bmod Q_1$. In this case, to compute the factorization of Q, we use the fact that all its irreducible factors have degrees that are multiples of n. For instance, if $d = 2$, then either Q is irreducible, or it has two factors of degree n. In this case, factorization takes expected time $O(\mathsf{C}(n)\log(n) + \mathsf{M}(n)\log(p))$ using Shoup (1994, Th. 26) and Gathen and Shoup (1992, Th. 5.4). More generally, for fixed d, a similar result holds, by trying all possible degrees for the factors of Q.

After possibly replacing Q by Q_1, and letting $m = \deg(Q)$, so that $m \leq nd$, we deduce that

$$\begin{array}{ccccc} \varphi : & \mathbb{A} = \mathbb{F}_p[X]/\langle P\rangle & \to & \mathbb{A}' = \mathbb{F}_p[X]/\langle Q\rangle \\ & X & \mapsto & T \end{array}$$

is a well-defined injection $\mathbb{F}_{p^n} \to \mathbb{F}_{p^m}$; besides, extending φ to a map $\mathbb{A}[Y] \to \mathbb{A}'[Y]$, we see that S is the root of $\varphi(A)$ we were looking for.

Once Q and T are known, applying φ to an element $B \in \mathbb{A}$ amounts to computing $B(T) \bmod Q$, and can thus be done in time $\mathsf{C}(n, m)$, which is $O(\mathsf{C}(n))$ for fixed d.

A special case. When $A(X, Y)$ has the form $Y^d - \alpha(X)$, and when we can take $Z = Y$, all computations can be done using univariate algorithms only.

Lemma 1. *Suppose that $A(X, Y) = Y^d - \alpha(X)$, for some $\alpha \in \mathbb{F}_p[X] - \{0\}$. Let $\rho \in \mathbb{F}_p[Y]$ be the minimal polynomial of α modulo P and let $Q \in \mathbb{F}_p[Y]$ be the minimal polynomial of Y modulo I. Then $Q = \rho(Y^d)$.*

Proof. It is enough to prove that the monic polynomials Q and $Q^\star = \rho(Y^d)$ have the same roots and are both squarefree. The roots of Q are the values y_i, for all (x_i, y_i) root of I in $\overline{\mathbb{F}_p}$, that is, for all (x_i, y_i) with $P(x_i) = 0$ and $y_i^d = \alpha(x_i)$; equivalently, these are the dth roots of the values $\alpha(x_i)$. The roots of ρ are the values $\alpha(x_i)$, where x_i are the roots of P, so the roots of $Q^\star$ are dth roots of the values $\alpha(x_i)$.

This proves the first claim; to establish the second one, note that Q is squarefree since I is a radical ideal (since $\alpha \neq 0$, the Jacobian matrix of (P, A) is invertible modulo I, and the Jacobian criterion implies radicality); ρ is irreducible with non-zero roots (again, since $\alpha \neq 0$), so $Q^\star = \rho(Y^d)$ has no repeated root. $\square$

Using the results of Subsection 2.1, computing ρ takes $O(\mathsf{C}(n))$ operations in $\mathbb{F}_p$. Knowing ρ, we deduce Q for free, and we can thus decide whether Y is a primitive element: this is the case if and only if $\deg(\rho) = n$. If not, we fall back on the general strategy. If Y is a primitive element, we can compute in time $O(\mathsf{C}(n))$ a polynomial $t \in \mathbb{F}_p[X]$ such that $X = t(\alpha) \bmod P$; then, we take $T = t(X^d)$ and $S = X$.

Compared to the general strategy, we save a factor of d in the degree of the extension we work with; in practice, the fact that we only use univariate arithmetic induces as well significant savings.

3. Computing χ modulo ℓ

In this section, we describe the computation of $\chi \bmod \ell$, for ℓ an odd prime. This is done exactly as in Schoof's algorithm in genus 1: we compute a description of the ℓ-torsion subgroup $\mathcal{J}[\ell]$, and use it to deduce $\chi \bmod \ell$ by means (mostly) of operations on univariate polynomials. A serious difficulty comes from the size of the objects we consider: $\mathcal{J}[\ell]$ has cardinality ℓ^4, so cases such as $\ell = 31$ are at the limit of what can be done as of now.

After describing the general strategy (Subsection 3.1), we describe the successive steps of the algorithm: resultant computations (Subsection 3.2), removal of parasite solutions (Subsection 3.3), and deduction of $\chi \bmod \ell$ (Subsection 3.4); we conclude with experimental results.

3.1. General strategy

Representing the ℓ-torsion. In this section, we will explain how to compute polynomials R, S, W, Z such that all elements in $\mathcal{J}[\ell] - \{0\}$ can be described as $\langle X^2 + u_1 X + u_0, v_1 X + v_0 \rangle$, where u_0, u_1, v_0, v_1 are the zeros of a triangular system of the form

$$\mathbf{I}_\ell \left| \begin{array}{rcl} V_0 & - & V_1\, Z(U_1) \\ V_1^2 & - & W(U_1) \\ U_0 & - & S(U_1) \\ & & R(U_1), \end{array} \right. \tag{2}$$

with R, S, W, Z in $\mathbb{F}_p[U_1]$, R squarefree of degree $(\ell^4 - 1)/2$, and S, W, Z of degrees less than $(\ell^4 - 1)/2$.

Note that the polynomials R, S, W, Z may not always exist; in what follows, we will assume that they do (this is expected to be the case in general). Let us make a few comments on this assumption. It requires first of all that all ℓ-torsion divisors have weight 2. Besides, if D is an ℓ-torsion divisor, then $-D$ is also an ℓ-torsion divisor; in Mumford representation, they share the same U-polynomials and have opposite V-polynomials.

Thus, our assumption also requires that all pairs of non-opposite ℓ-torsion divisors have distinct U_1-coordinates and that none of them should have $V_1 = 0$.

For a given curve, there exist infinitely many ℓ for which $\mathcal{J}[\ell]$ contains weight 1 divisors, contradicting the first requirement. On the other hand, we certainly expect this assumption to hold most of the time in practice, for the small values of ℓ we consider (this is indeed the case in our experiments).

A system encoding the ℓ-torsion. We follow the same strategy as in Gaudry and Harley (2000) and Gaudry and Schost (2004a, 2005), and write an ℓ-torsion divisor D as a sum $D = P_1 + P_2 - [2]\infty$, with P_1 and P_2 on $\mathcal{C}$; then, D is ℓ-torsion if $[\ell](P_1 - \infty) = -[\ell](P_2 - \infty)$. Cantor (1994) proved that there exist polynomials d_0, d_1, d_2, e_0, e_1, e_2 in $\mathbb{F}_p[X]$ such that for every point $P = (x, y)$ in $\mathcal{C}$, we have

$$[\ell](P - \infty) = \left\langle X^2 + \frac{d_1(x)}{d_2(x)}X + \frac{d_0(x)}{d_2(x)}, \frac{y}{e_2(x)}\left(e_1(x)X + e_0(x)\right)\right\rangle,$$

provided $d_2(x)e_2(x) \neq 0$ (of course, these polynomials depend on ℓ, but we rather not add an extra index). For ℓ odd and greater than 2, these polynomials have respective degrees $2\ell^2 - 1$, $2\ell^2 - 2$, $2\ell^2 - 3$, $3\ell^2 - 2$, $3\ell^2 - 3$ and $3\ell^2 - 2$ (if ℓ is even, then these polynomials all have a degree reduced by 5); these degrees can be deduced from the paper of Cantor (1994). For a fixed curve, and a given degree ℓ, the polynomials d_i and e_i can be computed in a number of field operations that is quasi-quadratic in ℓ, using recursion formulae by Cantor (or simply using the group law, keeping x and y as indeterminates). This adds a negligible contribution to the whole running time.

Let X_1, Y_1, X_2, Y_2 be indeterminates that represent the coordinates of P_1 and P_2. Taking coordinates in the equality $[\ell](P_1 - \infty) = -[\ell](P_2 - \infty)$, we obtain the system

$$\mathbf{E}\left|\begin{array}{rcl} E_1(X_1, X_2) & = & (d_1(X_1)d_2(X_2) - d_1(X_2)d_2(X_1))/(X_1 - X_2) = 0, \\ E_2(X_1, X_2) & = & (d_0(X_1)d_2(X_2) - d_0(X_2)d_2(X_1))/(X_1 - X_2) = 0, \\ F_1(X_1, X_2, Y_1, Y_2) & = & Y_1 e_1(X_1)e_2(X_2) + Y_2 e_1(X_2)e_2(X_1) = 0, \\ F_2(X_1, X_2, Y_1, Y_2) & = & Y_1 e_0(X_1)e_2(X_2) + Y_2 e_0(X_2)e_2(X_1) = 0, \end{array}\right.$$

together with $d_2(X_1)d_2(X_2)e_2(X_1)e_2(X_2)(X_1 - X_2) \neq 0$. Combining the third and fourth equations of $\mathbf{E}$, we get

$$e_0(X_1)e_1(X_2) - e_0(X_2)e_1(X_1) = 0.$$

Since we are looking for solutions such that $X_1 \neq X_2$, we are led to introduce the following new equation, which will be useful later on:

$$E_3(X_1, X_2) = (e_0(X_1)e_1(X_2) - e_0(X_2)e_1(X_1))/(X_1 - X_2).$$

Finally, we add the equations $Y_i^2 - f(X_i)$, to ensure that the points P_i are on $\mathcal{C}$.

One could want to fall back on generalist algorithms to solve the previous equations. However, we will not do so: these systems are extremely difficult to solve (in our cases, they could have millions of solutions), so it is necessary to develop ad-hoc solutions and to exploit any possible savings. In particular, this leads us to base our algorithms on a few experimental observations, offered without a proof.

As an illustration of this principle, we will actually forget about some of the inequations, by simply assuming that no solution of $\mathbf{E}$ cancels $d_2(X_1)d_2(X_2)e_2(X_1)e_2(X_2)$.

3.2. Computing an ideal contained in $\mathbf{I}_\ell$

The equations in $\mathbf{E}$ are symmetric under the permutation of (X_1, Y_1) and (X_2, Y_2). In Gaudry and Schost (2004a), we presented a way to take advantage of these symmetries and obtain an equivalent system in the symmetric coordinates X_1X_2 and $-X_1 - X_2$ (which are the coefficients of the U-polynomial of the divisor $D = P_1 + P_2 - [2]\infty$). We recall this approach and develop it further, starting with a discussion on rewriting techniques for some symmetric polynomials.

Handling symmetries. Let H be in $\mathbb{F}_p[X]$ and let X_1 and X_2 be the indeterminates introduced before. Then the *divided differences* of H are the bivariate symmetric polynomials

$$\begin{aligned} A_H &= \Big(H(X_1) - H(X_2)\Big)/(X_1 - X_2) \\ B_H &= \Big(X_1H(X_2) - X_2H(X_1)\Big)/(X_1 - X_2). \end{aligned}$$

Rewriting them in terms of the elementary symmetric polynomials, we let $\mathfrak{A}_H$ and $\mathfrak{B}_H$ be the unique polynomials in $\mathbb{F}_p[U_0, U_1]$ such that we have

$$\mathfrak{A}_H(X_1X_2, -X_1 - X_2) = A_H \quad \text{and} \quad \mathfrak{B}_H(X_1X_2, -X_1 - X_2) = B_H.$$

The following identity in $\mathbb{F}_p[U_0, U_1][X]$ can then be checked by verifying that it is true at the roots X_1 and X_2 of the polynomial $X^2 + U_1X + U_0$:

$$H = \mathfrak{A}_H X + \mathfrak{B}_H \mod (X^2 + U_1X + U_0). \tag{3}$$

Given H and $u_1 \in \mathbb{F}_p$, we will need below to compute $\mathfrak{A}_H(U_0, u_1)$ and $\mathfrak{B}_H(U_0, u_1)$. This problem amounts to reducing H modulo $X^2 + u_1X + U_0$ in $\mathbb{F}_p[U_0][X]$.

Our solution relies on polynomial shift. The main idea is to rewrite the polynomial $X^2 + u_1X + U_0$ as $(X + u_1/2)^2 - (u_1^2/4 - U_0)$. Let $K = H(X - u_1/2)$ in $\mathbb{F}_p[X]$. We group the coefficients of K according to the parity of their indices, forming the polynomials K_{odd} and K_{even} such that $K = K_{\text{even}}(X^2) + XK_{\text{odd}}(X^2)$. Since $H = K(X + u_1/2)$, this gives

$$H = K_{\text{even}}\left(\left(X + \frac{u_1}{2}\right)^2\right) + \left(X + \frac{u_1}{2}\right) K_{\text{odd}}\left(\left(X + \frac{u_1}{2}\right)^2\right).$$

Taking H modulo $X^2 + u_1X + U_0$, we have

$$H = K_{\text{even}}\left(\frac{u_1^2}{4} - U_0\right) + \left(X + \frac{u_1}{2}\right) K_{\text{odd}}\left(\frac{u_1^2}{4} - U_0\right) \mod (X^2 + u_1X + U_0). \tag{4}$$

Thus, computing $\mathfrak{A}_H(U_0, u_1)$ and $\mathfrak{B}_H(U_0, u_1)$ can be done by computing K by a polynomial shift, decomposing it into K_{even} and K_{odd}, applying another two polynomial shifts to get $K_{\text{even}}\left(u_1^2/4 - U_0\right)$ and $K_{\text{odd}}\left(u_1^2/4 - U_0\right)$, and concluding by means of (4). In view of what we recalled in Subsection 2.1 on polynomial shift, and assuming that $p > d$, with $d = \deg(H)$, we can compute $\mathfrak{A}_H(U_0, u_1)$ and $\mathfrak{B}_H(U_0, u_1)$ in $O(\mathsf{M}(d))$ operations.

Application. The previous equations E_1, E_2, E_3 can be rewritten in symmetric form using the previous construction: defining the polynomials

$$\begin{aligned} \mathfrak{E}_1 &= \mathfrak{A}_{d_1}\ \mathfrak{B}_{d_2} - \mathfrak{A}_{d_2}\ \mathfrak{B}_{d_1}, \\ \mathfrak{E}_2 &= \mathfrak{A}_{d_0}\ \mathfrak{B}_{d_2} - \mathfrak{A}_{d_2}\ \mathfrak{B}_{d_0}, \\ \mathfrak{E}_3 &= \mathfrak{A}_{e_0}\ \mathfrak{B}_{e_1} - \mathfrak{A}_{e_1}\ \mathfrak{B}_{e_0} \end{aligned}$$

in $\mathbb{F}_p[U_0, U_1]$, we have $E_i = \mathfrak{E}_i(X_1X_2, -X_1 - X_2)$ for $i = 1, 2, 3$. Thus, for any ℓ-torsion divisor $\langle X^2 + u_1X + u_0, v_1X + v_0\rangle$, (u_0, u_1) is a solution of $\mathfrak{E}_1 = \mathfrak{E}_2 = \mathfrak{E}_3 = 0$. In this subsection, we will describe how to solve the equations $\mathfrak{E}_1 = \mathfrak{E}_2 = 0$. We will discuss how to discard extraneous solutions in the next subsection.

We solve the system $\mathfrak{E}_1 = \mathfrak{E}_2 = 0$ by computing (factors of) the resultant $\tilde{r}(U_1) = \mathrm{res}(\mathfrak{E}_1, \mathfrak{E}_2, U_0)$ and the subresultant $\tilde{s}_0(U_1) + \tilde{s}_1(U_1)U_0 = \mathrm{sres}(\mathfrak{E}_1, \mathfrak{E}_2, U_0)$.

This is done using the algorithm of Subsection 2.1, by means of evaluation / interpolation at a geometric progression. Thus, given $u_1 \in \mathbb{F}_p$, we have to compute the polynomials $\mathfrak{E}_1(U_0, u_1)$ and $\mathfrak{E}_2(U_0, u_1)$; this boils down to computing the polynomials $\mathfrak{A}_{d_i}(U_0, u_1)$ and $\mathfrak{B}_{d_i}(U_0, u_1)$, for $i = 0, 1, 2$. In view of the result in the previous paragraph, for each value u_1, this can be done using $O(\mathsf{M}(\ell^2))$ operations in $\mathbb{F}_p$; this is less than the subsequent $O(\mathsf{M}(\ell^2)\log(\ell))$ incurred by the resultant computation.

Taking all required $O(\ell^4)$ values of u_1 into account, the total cost is $O(\ell^4\mathsf{M}(\ell^2)\log(\ell))$ operations in $\mathbb{F}_p$. This is not optimal, since the output of this step has size $O(\ell^4)$, but finding a better algorithm for this kind of resultant computation is a well-known open problem.

Parasites. Due to the very special form of the polynomials $\mathfrak{E}_1$ and $\mathfrak{E}_2$, there are predictable factors in $\tilde{r}$, $\tilde{s}_0$ and $\tilde{s}_1$, which generically do not correspond to solutions of the system $\mathfrak{E}_1 = \mathfrak{E}_2 = \mathfrak{E}_3 = 0$; in Gaudry and Schost (2004a), we called them *parasites*. We start by giving their precise form for $\tilde{r}$, $\tilde{s}_0$ and $\tilde{s}_1$, then explain how to exploit this information to save a constant factor in the running time.

Recall the definition of the polynomials SR and sr given in Subsection 2.1. Given the form of the polynomials $\mathfrak{E}_1$ and $\mathfrak{E}_2$, we expect $\rho = \mathrm{sr}(d_2)$ to occur as a factor of their resultant. Based on Cantor's recurrence formulae used to construct d_2, one can show that d_2 has the form $d_2 = f^3\delta^2$, where δ is a polynomial and f is the polynomial defining the curve $\mathcal{C}$. We can then deduce the following formula for ρ, which follows easily from the definition of the polynomials SR and sr:

$$\rho = f(X/2)^3\,\delta(X/2)\,\mathrm{SR}(f, \delta)^6\,\mathrm{sr}(\delta)^4\,\mathrm{sr}(f)^9.$$

The parasite factor for the subresultant coefficients $\tilde{s}_0$ and $\tilde{s}_1$ is more difficult to predict; experimentally, we observe that the following factor is always present in both of them:

$$\sigma = \mathrm{sr}(f)^4\,\mathrm{sr}(\delta)\,\mathrm{SR}(f, \delta)^2.$$

We use this definition in the implementation; if one is interested in a proven complexity result, one can always ignore these parasites, since taking them into account just changes the complexity by a constant factor.

We will be interested in computing $\tilde{R} = \tilde{r}/\rho$, $\tilde{S}_0 = \tilde{s}_0/\sigma$ and $\tilde{S}_1 = \tilde{s}_1/\sigma$. The previous formulae show that ρ has degree $2\ell^4 - 7\ell^2 + 6$, and σ has degree $(\ell^4 + \ell^2 - 10)/2$. After parasite removal, we observe that the degree of $\tilde{R}$ is about $2\ell^4$, and the degrees of $\tilde{S}_0$ and $\tilde{S}_1$ are about $7\ell^4/2$.

Being able to predict the parasites ρ and σ allows us to reduce the number of required evaluation points: for any given value $u_1 \in \mathbb{F}_p$, we compute $\tilde{r}(u_1)$, $\tilde{s}_0(u_1)$ and $\tilde{s}_1(u_1)$ by the subresultant algorithm, and separately $\rho(u_1)$ and $\sigma(u_1)$ using the algorithm in Subsection 2.1, for an extra cost of $O(\mathsf{M}(\ell^2))$; this gives us $\tilde{R}(u_1)$, $\tilde{S}_0(u_1)$ and $\tilde{S}_1(u_1)$. In view of the degrees of $\tilde{R}$, $\tilde{S}_0$ and $\tilde{S}_1$, we deduce that we need to do this for about $7\ell^4/2$ values of u_1. Without parasite prediction, we would need about $4\ell^4$ values, thus saving 12.5% (since we are only saving a constant factor, the cost remains $O(\ell^4\mathsf{M}(\ell^2)\log(\ell))$, with or without parasite prediction).

3.3. Refining to get $\mathbf{I}_\ell$

Given $\tilde{R}$, $\tilde{S}_0$ and $\tilde{S}_1$, we now show how to deduce $\mathbf{I}_\ell$ itself. We have to refine the set of solutions described by these polynomials, by discarding many extraneous solutions: $\tilde{R}$ has a degree that is about 4 times as large as the degree of the polynomial R we are looking for.

The direct approach would be to use the equations $\mathfrak{E}_1 = \mathfrak{E}_3 = 0$, apply again the resultant strategy, obtain another univariate polynomial that lies in $\mathbf{I}_\ell$, and take its GCD with $\tilde{R}$; we would then expect to obtain R. However, this second resultant will be at least as costly as the first one. Instead, we propose here two ways to refine the ideal, that both have a smaller time complexity, thus saving asymptotically a factor of 2 in the running time.

Using modular composition. We start by computing $\tilde{S} = -\tilde{S}_0/\tilde{S}_1 \bmod \tilde{R}$. We will then reintroduce the equation $\mathfrak{E}_3 \in \mathbb{F}_p[U_0, U_1]$ by computing $R' = \mathfrak{E}_3(\tilde{S}, U_1) \bmod \tilde{R}$, and replace $\tilde{R}$ by $\gcd(\tilde{R}, R')$.

The main question is to compute R' efficiently. In view of the definition of $\mathfrak{E}_3$, we see that R' is given by $\mathfrak{A}_{e_0}\,\mathfrak{B}_{e_1} - \mathfrak{A}_{e_1}\,\mathfrak{B}_{e_0}$, evaluated at $U_0 = \tilde{S}(U_1)$, and reduced modulo $\tilde{R}$. In other words, we have to compute $\mathfrak{A}_{e_i}(\tilde{S}, U_1) \bmod \tilde{R}$ and $\mathfrak{B}_{e_i}(\tilde{S}, U_1) \bmod \tilde{R}$, for $i = 0, 1$. Define the algebra

$$\tilde{\mathbb{B}} = \mathbb{F}_p[U_1, X]/\langle \tilde{R}, X^2 + U_1 X + \tilde{S}\rangle.$$

Equation (3) shows that $\mathfrak{A}_{e_i}(\tilde{S}, U_1) \bmod \tilde{R}$ and $\mathfrak{B}_{e_i}(\tilde{S}, U_1) \bmod \tilde{R}$ are respectively the coefficients of degree 1 and 0 in X of the remainder of $e_i(X)$ in $\tilde{\mathbb{B}}$. Computing this residue is a similar question to the reduction we saw in the previous subsection, and we could use a similar solution. However, this time, U_1 is kept as a variable; as a result, this approach would cost too much.

Instead, we use a bivariate modular composition, resulting in a cost $O(\mathsf{C}(\ell^2, \ell^4))$, which is $O(\ell\mathsf{M}(\ell^4) + \ell^{\omega+3})$; the memory requirement is $O(\ell^5)$ elements of $\mathbb{F}_p$.

Once we know R', we take its GCD with $\tilde{R}$; this takes a negligible $O(\mathsf{M}(\ell^4)\log(\ell))$ operations in $\mathbb{F}_p$. Experimentally, we observe that $\gcd(\tilde{R}, R')$ has degree $(\ell^4-1)/2$; this is thus the polynomial R we are looking for. Reducing $\tilde{S}$ modulo R, we get the polynomial S of $\mathbf{I}_\ell$, so it only remains to compute the polynomials W and Z. To this effect, we define the algebra (similar to $\tilde{\mathbb{B}}$)

$$\mathbb{B} = \mathbb{F}_p[U_1, X]/\langle R, X^2 + U_1 X + S\rangle.$$

From the equation $F_1 = 0$ of $\mathbf{E}$, we deduce that any solution (x_1, x_2, y_1, y_2) of $\mathbf{E}$ satisfies

$$y_1 y_2 = -y_2^2 \frac{e_1(x_2)e_2(x_1)}{e_1(x_1)e_2(x_2)}.$$

Since $y_2^2 = f(x_2)$, this can be rewritten as

$$y_1 y_2 = -f(x_2) \frac{e_1(x_2)e_2(x_1)}{e_1(x_1)e_2(x_2)}.$$

The V_1-coordinate of the weight-2 divisor $P_1 + P_2 - [2]\infty$ in Mumford representation is given by $\frac{y_1-y_2}{x_1-x_2}$; thus, its square equals

$$\frac{f(x_1) + f(x_2) - 2y_1y_2}{x_1^2 + x_2^2 - 2x_1x_2},$$

which can be expressed in terms of x_1, x_2 only using the former expression for y_1y_2. To obtain the polynomial $W(U_1)$, we evaluate the resulting expression at $x_1 = X$ and $X_2 = -U_1 - X$ in $\mathbb{B}$. The main cost comes from the computation of $e_1(X)$, $e_2(X)$, $e_1(-U_1 - X)$ and $e_2(-U_1 - X)$ in $\mathbb{B}$, which we do using modular composition as before, for a similar cost.

In the same spirit, we compute the last polynomial Z of $\mathbf{I}_\ell$, as the value of V_0/V_1, which we express in $\mathbb{B}$ as

$$\frac{V_0}{V_1} = \frac{x_1 f(x_2) - x_2 f(x_1)}{y_1 y_2 - f(x_2)}.$$

To summarize, the overall cost to get $\mathbf{I}_\ell$ using this technique is dominated by the modular compositions. Using the bivariate version of Brent and Kung's algorithm, this amounts to a number of operations of the form $O(\mathsf{C}(\ell^2, \ell^4))$, which is $O(\ell\mathsf{M}(\ell^4) + \ell^{\omega+3})$. The memory requirement is $O(\ell^5)$ elements of $\mathbb{F}_p$.

Using the group law. The algorithm based on modular composition is faster than the first step of computing $\tilde{R}$ and $\tilde{S}$. However, the memory requirement is $O(\ell)$ times larger, and can become the main limitation.

An alternative method is to build a "candidate" ℓ-torsion divisor $\tilde{D}_\ell$, with coefficients in an algebra that extends the algebra $\tilde{\mathbb{B}}$ defined above. This divisor $\tilde{D}_\ell$ is then multiplied by ℓ. Since the ideal used to construct it is smaller than $\mathbf{I}_\ell$, this does not give the zero divisor in the "Jacobian" over $\tilde{\mathbb{B}}$. Taking the GCD of $\tilde{R}$ and the denominators that occur in the result, we observe experimentally that we obtain the exact polynomial R in $\mathbf{I}_\ell$.

Let us give a few more details on the techniques we use to construct $\tilde{D}_\ell$. We start again with the algebra

$$\tilde{\mathbb{B}} = \mathbb{F}_p[U_1, X]/\langle \tilde{R}, X^2 + U_1 X + \tilde{S} \rangle.$$

Then, the abscissae X_1 and X_2 of the two points P_1 and P_2 defining $\tilde{D}_\ell$ are expressed in $\tilde{\mathbb{B}}$ as X and $-U_1 - X$. Their ordinates are defined in a degree-2 extension of $\tilde{\mathbb{B}}$, but using a strategy explained with more details in the next subsection, we will be able to perform Jacobian arithmetic with P_1 and P_2 at almost the same cost as if they were indeed defined over $\tilde{\mathbb{B}}$.

Computing $[\ell](P_1 - \infty)$ and $[\ell](P_2 - \infty)$, we deduce the squares of the V_1-coordinates of these divisors; they should be equal if $\tilde{D}_\ell = P_1 + P_2 - [2]\infty$ was indeed an ℓ-torsion element. In fact, their difference δ is a multiple of a factor of $\tilde{R}$; we observe experimentally that the GCD of δ and $\tilde{R}$ is the polynomial R in $\mathbf{I}_\ell$. In the same spirit, all other elements of $\mathbf{I}_\ell$ can be recovered with a constant number of additional operations in $\tilde{\mathbb{B}}$.

The overall cost to refine the ideal and get $\mathbf{I}_\ell$ is $O(\mathsf{M}(\ell^4)\log(\ell))$ operations in $\mathbb{F}_p$, with a memory requirement of $O(\ell^4)$ elements of $\mathbb{F}_p$. Indeed, one addition or multiplication in $\tilde{\mathbb{B}}$ uses $O(\mathsf{M}(\ell^4))$ operations in $\mathbb{F}_p$, and the multiplication of P_1 and P_2 by ℓ requires $O(\log(\ell))$ such operations; the subsequent GCD computations take time $O(\mathsf{M}(\ell^4)\log(\ell))$ as well.

3.4. Finding χ mod ℓ

Given $\mathbf{I}_\ell$, we describe next how to recover χ mod ℓ. In Gaudry and Schost (2004a), we factored the polynomial R defining $\mathbf{I}_\ell$; following Pitcher (2009), we avoid factorization, as it may actually become a bottleneck.

Let $\mathbb{D}$ be the residue class ring

$$\mathbb{D} = \mathbb{F}_p[U_1, U_0, V_1, V_0]/\langle R(U_1), U_0 - S(U_1), V_1^2 - W(U_1), V_0 - V_1 Z(U_1)\rangle.$$

Although $\mathbb{D}$ is in general not a field, but a product of fields, we may still define a "divisor" with coordinates in $\mathbb{D}$; in particular, we will write

$$D_\ell = \langle X^2 + U_1 X + U_0, V_1 X + V_0\rangle = \langle X^2 + U_1 X + S(U_1), V_1(X + Z(U_1))\rangle.$$

Applying a power of the Frobenius π to such a divisor is straightforward. We will also want to add these divisors, using the standard addition formulae. Since the group law in the Jacobian involves inversions, the possibility exists of a division by a zero-divisor. If this is the case, we obtain a factorization of R, and we dynamically switch to working modulo all factors of R separately; this does not hurt the complexity, or the practical runtime. Thus, one operation in the Jacobian with coordinates in $\mathbb{D}$ takes $O(\mathsf{M}(\ell^4)\log(\ell))$ operations in $\mathbb{F}_p$, and one application of π takes $O(\mathsf{M}(\ell^4)\log(p))$ operations in $\mathbb{F}_p$.

The algorithm. Since for all divisors D in $\mathcal{J}[\ell]$ we have

$$\pi^4(D) - [s_1 \bmod \ell]\pi^3(D) + [s_2 \bmod \ell]\pi^2(D) - [ps_1 \bmod \ell]\pi(D) + [p^2 \bmod \ell]D = 0,$$

we deduce the equality over $\mathbb{D}$

$$\pi^4(D_\ell) - [s_1 \bmod \ell]\pi^3(D_\ell) + [s_2 \bmod \ell]\pi^2(D_\ell) - [ps_1 \bmod \ell]\pi(D_\ell) + [p^2 \bmod \ell]D_\ell = 0.$$

To find all possible values of (s_1, s_2) in $(\mathbb{Z}/\ell\mathbb{Z})^2$ that satisfy this relation, we proceed as usual. We first compute the images $\pi^i(D_\ell)$ for $i = 1, 2, 3, 4$, and split the characteristic polynomial equality in a left hand side that involves s_1, and a right hand side that involves s_2:

$$\pi^4(D_\ell) + [p^2 \bmod \ell]D_\ell - [s_1 \bmod \ell](\pi^3(D_\ell) - [p \bmod \ell]\pi(D_\ell)) = -[s_2 \bmod \ell]\pi^2(D_\ell). \quad (5)$$

All possible left-hand-sides can be computed with $O(\ell)$ additions in the Jacobian. Then, using $O(\ell)$ additional operations in the Jacobian, all the right-hand sides can be computed and checked against the stored left-hand sides. The set of (s_1, s_2) modulo ℓ for which (5) holds can therefore be computed using $O(\ell)$ operations in $\mathbb{D}$, plus $O(1)$ applications of the Frobenius to elements of $\mathbb{D}$. The total is thus $O(\ell\mathsf{M}(\ell^4)\log(\ell) + \mathsf{M}(\ell^4)\log(p))$ operations in $\mathbb{F}_p$.

In general, only one pair (s_1, s_2) should remain, but in some cases there are several candidates. This can be dealt with, as we explain now.

To each such pair, one can associate a polynomial of degree 4 that annihilates the matrix of the Frobenius endomorphism acting on $\mathcal{J}[\ell]$. Therefore, each pair corresponds to a multiple of the minimal polynomial μ_ℓ of this endomorphism. Taking the GCD of all the polynomials constructed this way gives a multiple M_ℓ of μ_ℓ.

We will show that for all possible cases, we can deduce the right choice for (s_1, s_2) from M_ℓ. Remark first that M_ℓ is the GCD of all polynomials that annihilate the Frobenius endomorphism and whose roots in $\overline{\mathbb{F}_\ell}$ come in pairs $(\alpha, p/\alpha)$. Then, the conclusion follows from considering the following cases.

- If μ_ℓ has degree 1, it has the form $\mu_\ell = T - \alpha$, with $\alpha^2 = p$. Then $M_\ell = (T - \alpha)^2$ and one deduces that $\chi = M_\ell^2$.
- If μ_ℓ has degree 2 and a double root, then $\mu_\ell = (T - \alpha)^2$, with $\alpha^2 = p$. Again, $M_\ell = (T - \alpha)^2$ and $\chi = M_\ell^2$.

- If μ_ℓ has degree 2 and two distinct roots, there are two sub-cases: if the product of the two roots of μ_ℓ is different from p, then M_ℓ has degree 4, i.e. there is only one solution (s_1, s_2). Else, M_ℓ is equal to μ_ℓ, and again $\chi = M_\ell^2$.
- The last case is when μ_ℓ has degree 3. Then either M_ℓ has degree 3 as well (so $\mu_\ell = M_\ell$), and we complete it to χ using the fact that the constant term must be p^2, or M_ℓ has degree 4 and there is nothing to do.

To summarize, if $\deg(M_\ell) = 2$, we have $\chi = M_\ell^2$; if $\deg(M_\ell) = 3$, we have $\chi = (T + p^2/c)M_\ell$, where c is the constant term of M_ℓ; if $\deg(M_\ell) = 4$, we have $\chi = M_\ell$. As a consequence, in all cases, we can uniquely deduce the characteristic polynomial χ modulo ℓ.

Practical improvements. As a first obvious remark, in the former algorithm, one should not store all left-hand sides, but only their images by a hash function.

Secondly, we discuss how to avoid working over $\mathbb{D}$. The natural way to construct and work with D_ℓ is indeed to take coefficients in $\mathbb{D}$. However, it is possible to modify the group law in order not to have to work in $\mathbb{D}$, but only in $\mathbb{A} = \mathbb{F}_p[U_1]/\langle R\rangle$: even though the modified group law is slightly more expensive, this is a useful improvement, since arithmetic operations in $\mathbb{D}$ are three times as expensive as arithmetic operations in $\mathbb{A}$.

Remark that we can write D_ℓ as

$$D_\ell = \left\langle X^2 + U_1 X + S, \sqrt{W}(X + Z) \right\rangle,$$

where S, W and Z are in $\mathbb{A}$. Since all the divisors we need to manipulate are generated by Galois conjugates of D_ℓ, all of them can be represented by a 4-tuple of coordinates (F_0, F_1, G_0, G_1) in $\mathbb{A}$, such that the corresponding Mumford representation is

$$\left\langle X^2 + F_1 X + F_0, \sqrt{W}(G_1 X + G_0) \right\rangle.$$

When doubling or adding divisors represented by such a 4-tuple, one can express the result with a similar 4-tuple, through small modifications of the group law. Deriving the modified group law is easy from the formulae given for instance in Lange (2005): it suffices to replace V_0, V_1 by $\sqrt{W}G_0, \sqrt{W}G_1$ in the formulae and keep track of what they become. The Frobenius action can also be made to preserve this representation.

Thus, even though we are working in $\mathbb{D} = \mathbb{A}[\sqrt{W}]$, only 4 coordinates in $\mathbb{A}$ are required to give the Mumford representation of elements that would in principle be defined over $\mathbb{D}$. The cost is the same as that of classical Jacobian arithmetic over the algebra $\mathbb{A}$ of degree $(\ell^4 - 1)/2$, plus an additional half-a-dozen multiplications in $\mathbb{A}$ per Jacobian operation, in order to take into account the modified group law.

3.5. Summary and experimental results

We conclude this section by a summary of the former algorithm, followed by experimental results.

Input: an odd prime ℓ

Output: (s_1, s_2) mod ℓ

(1) compute $d_0, d_1, d_2, e_0, e_1, e_2 \in \mathbb{F}_p[X]$

(2) compute $\tilde{R} = \tilde{r}/\rho$, $\tilde{S}_0 = \tilde{s}_0/\sigma$ and $\tilde{S}_1 = \tilde{s}_1/\sigma$ by evaluation / interpolation, with

$$\tilde{r}(U_1) = \mathrm{res}(\mathfrak{E}_1, \mathfrak{E}_2, U_0), \quad \tilde{s}_0(U_1) + \tilde{s}_1(U_1)U_0 = \mathrm{sres}(\mathfrak{E}_1, \mathfrak{E}_2, U_0)$$

and

$$\rho = f(X/2)^3\, \delta(X/2)\, \mathrm{SR}(f,\delta)^6\, \mathrm{sr}(\delta)^4\, \mathrm{sr}(f)^9, \quad \sigma = \mathrm{sr}(f)^4\, \mathrm{sr}(\delta)\, \mathrm{SR}(f,\delta)^2.$$

(3) compute R, S, W, Z by modular composition or using the group law (Subsection 3.3)
(4) find all possible pairs $(s_1, s_2) \in (\mathbb{Z}/\ell\mathbb{Z})^2$ that satisfy (5)
(5) deduce the characteristic polynomial χ mod ℓ (Subsection 3.4) and the actual value of (s_1, s_2) mod ℓ

Finally, we give running times for prime values of ℓ from 5 to 31. The latest cases become quite challenging, from the memory and running time points of view: we compute resultants, and take GCDs, in degrees more than a million.

In Table 1, we give detailed timings (in seconds) for the values of ℓ we are interested in; timings are measured on one core of a Xeon L5640 at 2.27 GHz. We used our NTL-based implementation, running on a typical genus 2 curve defined over $\mathbb{F}_p$, where $p = 2^{127} - 1$. We give the time for resultants (first, for 1000 specialized resultants, then for all the ones we need), for refining to get $\mathbf{I}_\ell$ (comparing the two strategies of Subsection 3.3), computing the Frobenius $\pi^i(D_\ell)$, and finally finding the values of (s_1, s_2) mod ℓ. To summarize, dealing with $\ell = 31$ requires about 10 CPU days.

The cost of computing all resultants is of course the dominant one, but this step is easily parallelizable and requires almost no memory. The refining step, which is not parallelized, can become the bottleneck, especially in terms of memory.

The approach using the group law is asymptotically the best, both from the time and space point of view. However, the constant hidden by the big-O is very high: for the current sizes, the group law method has no interest in terms of running time. However, the algorithm using modular composition uses more memory: as soon as $\ell \geq 29$, the computation does not fit anymore in 8 GB of RAM; by contrast, the group law method allows us to deal with $\ell = 31$ in this amount of RAM.

For our large-scale computation described in Section 5, we re-implemented the group law approach in C, using the Mpfq library (Gaudry and Thomé, 2007), in order to take full advantage of the particular form of the prime. This also saves some memory.

4. Lifting torsion elements of index ℓ^k

In this section, we explain how to compute torsion divisors of index ℓ^k, for ℓ in $\{2, 3, 5, 7\}$, and use them in the point-counting algorithm. In all that follows, ℓ will be a prime different from p (as is clear from the restricted list of values of ℓ we consider).

The general process is as follows: for a given value of ℓ, we start by computing a divisor P_1 in $\mathcal{J}[\ell]$. Explicitly, using the algorithm of the previous section, we can compute all ℓ-torsion divisors; then, after factoring the polynomial R of the triangular system $\mathbf{I}_\ell$, we take for P_1 an element of $\mathcal{J}[\ell]$ of smallest degree over $\mathbb{F}_p$ (some other considerations actually come into play; they are discussed below). Knowing P_1, we determine a sequence of torsion divisors P_k, with $P_k = [\ell]P_{k+1}$ (so P_k is in $\mathcal{J}[\ell^k]$). At each step, knowing P_k allows us to deduce some information about (s_1, s_2). We continue as far as feasible.

In Subsections 4.1 and 4.2, we will give more details on this process: roughly speaking, we will prove that one may expect P_k to be defined in degree $e_k \approx \ell^k$ and that knowing P_k gives us (s_1, s_2) modulo $\ell^{k-\kappa}$, for some integer κ.

ℓ	resultant		refine to get $\mathbf{I}_\ell$		Frobenius	get (s_1, s_2)
	1000 resultants	all resultants	ModComp	group law		
5	2.11	3.80	2.27	8.61	6.96	1.41
7	4.89	37.3	13.6	68.7	30.1	6.79
11	17.6	867	119	471	154	49.8
13	29.3	2850	297	1250	318	216
17	57.9	16700	1480	3670	1250	982
19	74.1	33400	2120	6080	1490	1180
23	131	127000	8890	20000	5100	5620
29	210	517000	27000	68600	11800	17100
31	229	737000	34300	84700	12300	19100

Table 1. Details for ℓ-torsion. Timings are given in seconds on a single core of a Xeon L5640 processor at 2.27 GHz.

Computationally, the essential difficulty is the construction of the sequence P_k: going from P_k to P_{k+1} involves a "division by ℓ" in the Jacobian, which requires solving a system of polynomial equations. This will be the main part of this section: Subsections 4.3–4.5 describe our solutions for $\ell = 2$, $\ell = 3$, and $\ell = 5$ or 7, which all take quite different forms.

4.1. Overview

For all values of ℓ, our approach is the same: starting from $P_1 \in \mathcal{J}[\ell]$, we construct $P_2, P_3, \ldots$ such that $P_k = [\ell]P_{k+1}$ holds for all $k \geq 1$. Let e_k be the degree of the field of definition of P_k over $\mathbb{F}_p$. Lemma 6 (proved in the next subsection) shows that $e_{k+1} \leq \ell e_k$. Since the points of $\mathcal{J}[\ell]$ live in an extension of $\mathbb{F}_p$ of degree bounded by ℓ^4, we deduce that $e_k \leq \ell^{k+3}$.

We will always assume that every P_k has weight 2. Letting $F \in \mathbb{F}_p[T]$ be an irreducible polynomial of degree e_k, so that $\mathbb{F}_{p^{e_k}} = \mathbb{F}_p[T]/\langle F \rangle$, the divisor $P_k = \langle X^2 + u_1 X + u_0, v_1 X + v_0 \rangle$ will thus be described by means of polynomials of the form

$$\mathbf{C}_{\ell^k} \left| \begin{array}{rcl} v_0 & = & Z(T) \\ v_1 & = & W(T) \\ u_0 & = & S(T) \\ u_1 & = & R(T) \\ F(T) & = & 0, \end{array} \right. \tag{6}$$

with R, S, W, Z in $\mathbb{F}_p[T]$. Remark that one Jacobian operation with any divisor defined over $\mathbb{F}_{p^{e_k}}$ takes $O(\mathsf{M}(e_k)\log(e_k))$ operations in $\mathbb{F}_p$.

Knowing $\mathbf{C}_{\ell^k}$, we look for (s_1, s_2) that satisfy the relation

$$\pi^4(P_k) - [s_1]\pi^3(P_k) + [s_2]\pi^2(P_k) - [ps_1]\pi(P_k) + [p^2]P_k = 0. \tag{7}$$

Since P_k is in $\mathcal{J}[\ell^k]$, the best we can hope for is to obtain $(s_1, s_2) \bmod \ell^k$. We do not quite obtain this: when $\mathcal{J}$ is absolutely simple, Lemma 4 below will prove that the former relation uniquely determines $(s_1, s_2) \bmod \ell^{k-\kappa}$, for some integer κ.

To find $(s_1, s_2) \bmod \ell^{k-\kappa}$, we proceed as in Subsection 3.4, and rewrite (7) as

$$\pi^4(P_k) + [p^2]P_k - [s_1](\pi^3(P_k) + [p]\pi(P_k)) = -[s_2]\pi^2(P_k).$$

Assuming that we know $(s_1, s_2) \bmod \ell^{k-\kappa-1}$ from previous steps, we have ℓ choices to test for s_1 and s_2; all possible choices differ by multiples of $\ell^{k-\kappa-1}$. Thus, we need to precompute $[\ell^{k-\kappa-1}](\pi^3(P_k)+[p]\pi(P_k))$ and $[\ell^{k-\kappa-1}]\pi^2(P_k)$: this requires $O(1)$ Frobenius computations, and $O(k)$ operations in the Jacobian, for a total of $O(k\mathsf{M}(e_k)\log(e_k) + \mathsf{M}(e_k)\log(p))$ operations in $\mathbb{F}_p$ (we assume that ℓ is fixed in these cost estimates). Finding $(s_1, s_2) \bmod \ell^{k-\kappa}$ takes another $O(\ell) = O(1)$ Jacobian operations, which is negligible.

To initiate the next step, we need to compute $\mathbf{C}_{\ell^{k+1}}$; this amounts to solving polynomial equations for the coordinates of P_{k+1}. We do not explain this in detail here: this is the object of the last subsections, with different solutions for the values of ℓ we consider. In all cases, the cost is an expected $O(\mathsf{C}(e_k)\log(e_k) + \mathsf{M}(e_k)\log(p))$ operations in $\mathbb{F}_p$. This is the dominant step; the constants hidden in the big-O grow (quickly) with ℓ, and a lot of care is put in finding the most efficient solution.

4.2. A few useful lemmas

In this subsection, we prove some results that were claimed before, on the information we can deduce from P_k about (s_1, s_2), and on the field of definition of P_k.

Since P_k is in $\mathcal{J}[\ell^k]$, one would expect that it determines (s_1, s_2) modulo ℓ^k. There are two obstructions to this: first, P_k and its conjugates might not generate the whole $\mathcal{J}[\ell^k]$; second, testing the possible annihilating polynomials for P_k gives information only on the minimal polynomial of π, not on its characteristic polynomial. We will show that under some mild conditions, these two obstructions introduce only a constant shift, as announced in the preamble: for k large enough, P_k completely determines (s_1, s_2) modulo $\ell^{k-\kappa}$, for some constant κ that depends on the sequence P_k.

We remark that this phenomena does not exist for elliptic curves: assume that P_k is a point in $E[\ell^k] \setminus E[\ell^{k-1}]$, where E is an elliptic curve defined over $\mathbb{F}_p$. If there are two candidate integers s and s' such that $\pi^2(P) - [s]\pi(P) + [p]P = 0$, and such that the same equality holds for s', then subtracting the two equations, one gets $[s - s']\pi(P) = 0$, so that $[s - s']P = 0$ and therefore s and s' are congruent modulo ℓ^k. It is also interesting to note that these obstructions also vanish in genus 2 in the case where ℓ does not divide the discriminant of the characteristic polynomial of π (see Remark 5 below), which is the typical case for large ℓ. Since in this section we are concerned with small values of ℓ, they often divide the discriminant and we cannot ignore the obstructions.

In what follows, we let $T_\ell(\mathcal{J})$ be the Tate module of degree ℓ. We consider a fixed $\mathbb{Z}_\ell$-basis (E_1, E_2, E_3, E_4) of $T_\ell(\mathcal{J})$, and we denote by τ the matrix of the Frobenius endomorphism π in this basis. The determinant of τ is equal to p^2 and is therefore invertible in $\mathbb{Z}_\ell$, so that the matrix τ is invertible as well.

The first step is to prove that for a good choice of the sequence P_k, there exists $k_0 \geq 1$ such that for $k \geq k_0$, P_k and all its conjugates generate $\mathcal{J}[\ell^{k-k_0+1}]$: up to the loss of precision induced by k_0, this will imply that a characteristic polynomial equality for P_k will induce a similar equality for all of $\mathcal{J}[\ell^{k-k_0+1}]$.

Unfortunately, this claim is not true in general: for instance, if the Jacobian splits as a product of two isomorphic elliptic curves, then the action of the Frobenius on $T_\ell(\mathcal{J})$ is block-diagonal, with identical invariants on both blocks. In this case, there is no element whose conjugates can generate the whole ambient space. Thus, in all that follows, *we will suppose that $\mathcal{J}$ is absolutely simple.*

Lemma 2. *There exists an integer $k_0 \geq 1$ and $\mathcal{P}$ in $\mathcal{J}[\ell^{k_0}]$ such that $\mathcal{J}[\ell]$ is contained in the subgroup generated by $\mathcal{P}$ and its conjugates.*

Proof. Let $V_\ell(J)$ be $T_\ell(J) \otimes_{\mathbb{Z}_\ell} \mathbb{Q}_\ell$, which is a $\mathbb{Q}_\ell$-vector space of dimension 4. Since $\mathcal{J}$ is absolutely simple, the characteristic polynomial of π is irreducible over $\mathbb{Q}$, and therefore has no multiple factor over $\mathbb{Q}_\ell$. This implies that the characteristic and the minimal polynomials of π are equal, and therefore there exists a basis of $V_\ell(J)$ such that the matrix of π in this basis is a companion matrix. Any element $\hat{\mathcal{P}}$ of this basis is such that its conjugates generate the whole space $V_\ell(J)$.

Without loss of generality one can assume furthermore that $\hat{\mathcal{P}}$ has coefficients in $\mathbb{Z}_\ell$, and therefore belongs to $T_\ell(\mathcal{J})$. Since τ also has entries in $\mathbb{Z}_\ell$, the coordinate vectors of the family $(\hat{\mathcal{P}}, \tau\hat{\mathcal{P}}, \tau^2\hat{\mathcal{P}}, \tau^3\hat{\mathcal{P}})$ in the basis (E_1, E_2, E_3, E_4) give a matrix A with coefficients in $\mathbb{Z}_\ell$; its non-zero determinant is therefore in $\mathbb{Z}_\ell$ as well. Let k_0 be such that the valuation of this determinant is $k_0 - 1$.

We consider the point $\mathcal{P}$ obtained by projecting $\hat{\mathcal{P}}$ modulo ℓ^{k_0}; hence, $\mathcal{P}$ is in $\mathcal{J}[\ell^{k_0}]$. We will show that $\mathcal{J}[\ell]$ can be generated by $\mathcal{P}$ and its conjugates.

Let $(B_1, B_2, B_3, B_4) \in \mathcal{J}[\ell^{k_0}]^4$ be obtained by reducing (E_1, E_2, E_3, E_4) modulo ℓ^{k_0}. Since these divisors form a basis of $\mathcal{J}[\ell^{k_0}]$, any Q in $\mathcal{J}[\ell]$ can be written as a combination of (B_1, B_2, B_3, B_4). Besides, since Q is ℓ-torsion, all its coordinates are divisible by ℓ^{k_0-1}, so we have $Q = \sum[q_i\ell^{k_0-1}]B_i$, where q_i are defined modulo ℓ. Consider the inverse matrix of A over $\mathbb{Q}_\ell$; since the valuation of the determinant of A is $k_0 - 1$, its inverse has entries that become integers after multiplication by ℓ^{k_0-1}. Let further q be the vector of entries $(q_i\ell^{k_0-1})$. Then, the vector $v = A^{-1}q$ has entries in $\mathbb{Z}_\ell$ and answers the question: projecting the equation $Av = q$ modulo ℓ^{k_0} gives a combination of conjugates of $\mathcal{P}$ that equals Q. □

The main property of $\mathcal{P}$ is that, together with its conjugates, it generates $\mathcal{J}[\ell]$. The following lemma proves that dividing by ℓ propagates this property to higher level torsion subgroups.

Lemma 3. *Let $k \geq 1$ and let $P \in \mathcal{J}$ be such that $\mathcal{J}[\ell^k]$ is contained in the subgroup generated by P and its conjugates. Then for any $Q \in \mathcal{J}$ such that $P = [\ell]Q$, $\mathcal{J}[\ell^{k+1}]$ is contained in the subgroup generated by Q and its conjugates.*

Proof. Let Q' be in $\mathcal{J}[\ell^{k+1}]$. Since $[\ell]Q'$ is in $\mathcal{J}[\ell^k]$, it can be expressed as a linear combination of the conjugates of P, so we have $[\ell]Q' = \sum_i[\lambda_i]\pi^i(P)$, where λ_i are integers. Replacing P by $[\ell]Q$, we get $[\ell]Q' = [\ell]\sum_i[\lambda_i]\pi^i(Q)$. Hence, $Q' - \sum_i[\lambda_i]\pi^i(Q)$ is in $\mathcal{J}[\ell]$, so Q' is in the group generated by the conjugates of Q, up to an ℓ-torsion element. Finally, since $\mathcal{J}[\ell]$ is generated by the conjugates of P, it is also generated by conjugates of Q. □

From now on, we will assume that the sequence (P_k) constructed by successive division by ℓ in the Jacobian is such that for some k_0, the divisor P_{k_0} and its conjugates generate $\mathcal{J}[\ell]$. Lemma 2 ensures that such a divisor P_{k_0} exists, and by Lemma 3, for all $k \geq k_0$, P_k and its conjugates generate $\mathcal{J}[\ell^{k-k_0+1}]$.

Assuming we have found a suitable sequence (P_k), we prove that given P_k, one can find (s_1, s_2), not exactly modulo ℓ^k, but at least modulo $\ell^{k-\kappa}$, for some fixed κ.

Lemma 4. *There exists an integer $\kappa \geq 0$ such that for any $k > \kappa$, the equality*

$$\pi^4(P_k) - [s_1]\pi^3(P_k) + [s_2]\pi^2(P_k) - [ps_1]\pi(P_k) + [p^2]P_k = 0$$

uniquely determines (s_1, s_2) modulo $\ell^{k-\kappa}$.

Proof. Since $\mathcal{J}$ is absolutely simple, the characteristic polynomial of τ is irreducible over $\mathbb{Q}$, and therefore it is squarefree over $\mathbb{Q}_\ell$. Hence the minimal polynomial of τ is equal to its characteristic polynomial and is of degree 4. Since τ does not satisfy a monic quadratic equation over $\mathbb{Q}_\ell$, there exists k_1, such that for all $k > k_1$, τ does not satisfy a monic quadratic equation modulo ℓ^k.

We can then prove the lemma, taking $\kappa = k_0 + k_1 - 1$. Suppose indeed that for any $k > k_1$, there exists (s_1, s_2) and (s_1', s_2') in $\mathbb{Z}_\ell^2$ such that we have simultaneously

$$\pi^4(P_k) - [s_1]\pi^3(P_k) + [s_2]\pi^2(P_k) - [ps_1]\pi(P_k) + [p^2]P_k = 0$$

and

$$\pi^4(P_k) - [s_1']\pi^3(P_k) + [s_2']\pi^2(P_k) - [ps_1']\pi(P_k) + [p^2]P_k = 0.$$

These characteristic polynomial equalities hold as well for all conjugates of P_k; since P_k and its conjugates generate $\mathcal{J}[\ell^{k-k_0+1}]$, this implies that we have

$$\tau^4 - s_1\tau^3 + s_2\tau^2 - ps_1\tau + p^2 = 0 \bmod \ell^{k-k_0+1}$$

and

$$\tau^4 - s_1'\tau^3 + s_2'\tau^2 - ps_1'\tau + p^2 = 0 \bmod \ell^{k-k_0+1}.$$

For simplicity, let $k' = k - k_0 + 1$. By subtraction, defining $a = s_1 - s_1'$ and $b = s_2 - s_2'$, we find

$$a\tau^3 - b\tau^2 + pa\tau = 0 \bmod \ell^{k'};$$

since τ is invertible modulo $\ell^{k'}$, this implies $a\tau^2 - b\tau + pa = 0 \bmod \ell^{k'}$. Let m be the ℓ-adic valuation of a and, let $\alpha = a/\ell^m$, so that $\ell^m\alpha\tau^2 + b\tau + p\ell^m\alpha = 0 \bmod \ell^{k'}$.

If $m \geq k'$, we deduce that $b\tau = 0 \bmod \ell^{k'}$; since τ is invertible, we get $a = b = 0 \bmod \ell^{k'}$, which is (stronger than) what we wanted to prove. Else, using again the invertibility of τ, we deduce that $b = 0 \bmod \ell^m$; letting $\beta = b/\ell^m$, we get $\alpha\tau^2 + \beta\tau + p\alpha = 0 \bmod \ell^{k'-m}$. Since α is invertible, the definition of k_1 implies that $k' - m \leq k_1$, or $m \geq k' - k_1$. This can be rewritten as $m \geq k - (k_0 + k_1 - 1)$. □

Generically, we expect that κ is small, and since ℓ is small as well, finding a suitable start for the sequence (P_k) can be done with some brute force approach. In our experiments, we computed all the ℓ-torsion, and ℓ^2-torsion when feasible (for $\ell \leq 3$), and we picked P_1 (and P_2, when feasible) of smallest degree among the choices for which the loss of precision was minimal. We always found a sequence with $\kappa \leq 3$.

Following a suggestion by one referee, let us also mention that if $\kappa > 0$, it could be possible to lift say two sequences (P_k) and (P_k'), and combine the information from both sequences. We have not attempted this yet.

Remark 5. In the case where the characteristic polynomial of π has no repeated factor modulo ℓ, there exists a sequence P_k such that the two obstructions disappear and for all k, the point P_k completely determines (s_1, s_2) modulo ℓ^k. For the first obstruction, we can follow the beginning of the proof of Lemma 2, replacing $V_\ell(\mathcal{J})$ by the $\mathbb{F}_\ell$-vector space $\mathcal{J}[\ell]$. Since π has no repeated factor modulo ℓ, there exists a basis of $\mathcal{J}[\ell]$ such that the matrix of π with respect to this basis is a companion matrix, and any vector of the basis is a valid P_1 yielding $k_0 = 1$. For the second obstruction, the proof of Lemma 4 is simplified by the fact that the minimal polynomial of τ modulo ℓ is of degree 4, and therefore one can take $k_1 = 0$, and finally $\kappa = 0$.

We finish this subsection with a study of the field of definition of P_k.

Lemma 6. *Let d be a positive integer such that the points of $\mathcal{J}[\ell]$ are defined over $\mathbb{F}_{p^d}$, and let $P \in \mathcal{J}$ be defined over $\mathbb{F}_{p^d}$ as well. Then any $Q \in \mathcal{J}$ such that $P = [\ell]Q$ is defined over $\mathbb{F}_{p^{\ell d}}$.*

Proof. From the equalities $\pi^d(P) = P$ and $P = [\ell]Q$, we deduce that $[\ell](\pi^d(Q) - Q) = 0$, so $\pi^d(Q) - Q$ is in $\mathcal{J}[\ell]$. Let us denote it by T, and observe that $\pi^d(T) = T$. By successive applications of π^d, it follows that $\pi^{(i+1)d}(Q) - \pi^i(Q) = T$ for all $i \geq 0$. Summing these equalities for $i = 0, \ldots, \ell - 1$ shows that $\pi^{\ell d}(Q) = Q$. □

Recall that e_k is the degree of the field of definition of P_k over $\mathbb{F}_p$. Since for $k \geq k_0$, P_k and its conjugates generate $\mathcal{J}[\ell^{k-k_0+1}]$, and in particular $\mathcal{J}[\ell]$, we deduce that for $k \geq k_0$, the points of $\mathcal{J}[\ell]$ are defined over $\mathbb{F}_{p^{e_k}}$. The former lemma then implies that either $e_{k+1} = e_k$ or $e_{k+1} = \ell e_k$.

Finally, we prove that for k large enough, we are in the case $e_{k+1} = \ell e_k$. The following claim is similar to Lercier (1997, Cor. 4) and Feo (2010, Prop. 5), which hold in the elliptic case, when $\ell = p$. It proves that the degree d_k of the field of definition of the points of $\mathcal{J}[\ell^k]$ satisfies $d_{k+1} = \ell d_k$ for k large enough. Since, for $k \geq k_0$, P_k and its conjugates generate $\mathcal{J}[\ell^{k-k_0+1}]$, we deduce the inequality $d_{k-k_0+1} \leq e_k$, which implies that $e_{k+1} = \ell e_k$ for k large enough.

Lemma 7. *For $k \geq 1$, let d_k be the smallest integer such that the points of $\mathcal{J}[\ell^k]$ are defined over $\mathbb{F}_{p^{d_k}}$. Then for k large enough, we have $d_{k+1} = \ell d_k$.*

Proof. We prove that for all $k \geq 1$, either $d_{k+1} = \ell d_k$ or $d_1 = d_2 = \cdots = d_{k+1}$; this is sufficient to establish our claim.

Let τ_k be the matrix obtained from τ by projecting each entry in $\mathbb{Z}/\ell^k\mathbb{Z}$; then, the matrix τ_k is invertible. Since π generates the Galois group of $\overline{\mathbb{F}_p}$ over $\mathbb{F}_p$, the extension degree d_k is the order of τ_k in the group of invertible matrices over $\mathbb{Z}/\ell^k\mathbb{Z}$.

The matrix $\tau_k^{d_k}$ is the identity matrix in $\mathbb{Z}/\ell^k\mathbb{Z}$, so we will write $\tau^{d_k} = I + \alpha_k \ell^k$, where I is the identity matrix and α_k is a matrix with coefficients in $\mathbb{Z}_\ell$. Remark that $d_{k+1} = d_k$ if and only if $\alpha_k = 0 \bmod \ell$.

Taking ℓth power, we deduce that $\tau^{\ell d_k} = (I + \alpha_k \ell^k)^\ell$, and thus $\tau^{\ell d_k} = I + \alpha_k \ell^{k+1} \bmod \ell^{k+2}$. A first consequence is that $\tau^{\ell d_k} = I \bmod \ell^{k+1}$; since d_k divides d_{k+1}, we get that d_{k+1} can be equal to either d_k or ℓd_k. Besides, if $d_{k+1} = \ell d_k$, we obtain that $\alpha_{k+1} = \alpha_k \bmod \ell$; in particular, since $\alpha_k \neq 0 \bmod \ell$, we deduce that $\alpha_{k+1} \neq 0 \bmod \ell$, and thus that $d_{k+2} = \ell d_{k+1}$. □

4.3. Lifting the 2^k-torsion

In this subsection, we take $\ell = 2$ and we explain how to compute the sequence (P_k) of 2^k-torsion divisors. Given P_k, P_{k+1} is obtained by solving the equation $P_k = [2]P_{k+1}$. We will actually forget that P_k is a 2^k-torsion divisor: given any divisor P, we will be interested in finding a divisor Q such that $P = [2]Q$.

There is one aspect in which the case $\ell = 2$ differs from the rest of our treatment: instead of working with Mumford coordinates in $\mathcal{J}$, we will work in the associated Kummer surface $\mathcal{K} \subset \mathbb{P}^3$, which is the quotient of $\mathcal{J}$ by the hyperelliptic involution. The Kummer surface is not a group, but doubling in $\mathcal{K}$ still makes sense; in general, $\mathcal{K}$ is endowed with what is usually called a *pseudo-group law*, that still allows for scalar multiplication. We refer to Gaudry (2007) for details, and for the formulae we will use below.

Taking as input the coordinates of the image of P in $\mathcal{K}$, we compute the coordinates of the image of Q in $\mathcal{K}$. The upside is that the simple doubling formulae for $\mathcal{K}$ allow for an efficient algorithm for division by 2 in $\mathcal{K}$, that uses only square root computations: almost all the work boils down to using the algorithm of Subsection 2.2. The counterpart is that the images of the divisors (P_k) in $\mathcal{K}$ need to be lifted back in $\mathcal{J}$ to find (s_1, s_2); this is however a mild problem, for which we refer again to Gaudry (2007).

Overview. The doubling formulae in the Kummer surface rely essentially on squarings. Given a point $(x : y : z : t)$ in $\mathcal{K} \subset \mathbb{P}^3$, its double $(\tilde{x} : \tilde{y} : \tilde{z} : \tilde{t}) = [2](x : y : z : t)$ is given by the following operations: we compute

$$\begin{aligned} x' &= x^2 + y^2 + z^2 + t^2 \\ y' &= x^2 + y^2 - z^2 - t^2 \\ z' &= x^2 - y^2 + z^2 - t^2 \\ t' &= x^2 - y^2 - z^2 + t^2 \end{aligned} \tag{8}$$

then

$$\begin{aligned} x'' &= x'^2 \\ y'' &= y'_0 y'^2 \\ z'' &= z'_0 z'^2 \\ t'' &= t'_0 t'^2 \end{aligned} \tag{9}$$

and finally

$$\begin{aligned} \tilde{x} &= (x'' + y'' + z'' + t'') \\ \tilde{y} &= y_0(x'' + y'' - z'' - t'') \\ \tilde{z} &= z_0(x'' - y'' + z'' - t'') \\ \tilde{t} &= t_0(x'' - y'' - z'' + t''); \end{aligned} \tag{10}$$

in these equations, y_0, z_0, t_0, y'_0, z'_0, t'_0 are constants that depends only on $\mathcal{K}$ and can be easily computed from the equation of $\mathcal{C}$.

Our question is then the following: given $(\tilde{x} : \tilde{y} : \tilde{z} : \tilde{t})$, we want to invert this map, that is, to find $(x : y : z : t)$ such that $[2](x : y : z : t) = (\tilde{x} : \tilde{y} : \tilde{z} : \tilde{t})$. Assuming

that $(\tilde{x} : \tilde{y} : \tilde{z} : \tilde{t})$ and all points of $\mathcal{J}[2]$ are defined over $\mathbb{F}_{p^e}$, Lemma 6 implies that $(x : y : z : t)$ is defined over $\mathbb{F}_{p^{e'}}$, with either $e' = e$ or $e' = 2e$.

Since the transformation from $(x'^2 : y'^2 : z'^2 : t'^2)$ to $(\tilde{x} : \tilde{y} : \tilde{z} : \tilde{t})$ is linear and easily invertible, we can assume that we know $(\alpha : \beta : \gamma : \delta) = (x'^2 : y'^2 : z'^2 : t'^2)$.

First, we recover $(x' : y' : z' : t')$. The point $(x : y : z : t)$ satisfies the defining equation of $\mathcal{K}$, which takes the form

$$(x^4+y^4+z^4+t^4)-F(x^2t^2+y^2z^2)-G(x^2z^2+y^2t^2)-H(x^2y^2+t^2z^2)+2Exyzt = 0, \quad (11)$$

for some constants E, F, G, H that can be computed from the equation of $\mathcal{C}$. One can then check that $(x' : y' : z' : t')$ satisfies a similar equation, of the form

$$\begin{aligned}
&(E-F-G-H+2)(E+F+G+H-2)x'^4\\
+&(E+F+G-H+2)(E-F-G+H-2)y'^4\\
+&(E+F-G+H+2)(E-F+G-H-2)z'^4\\
+&(E-F+G+H+2)(E+F-G-H-2)t'^4\\
&-2(-(F+G)^2+H(H-4)+E^2+4)x'^2y'^2\\
&-2(-(G+H)^2+F(F-4)+E^2+4)x'^2t'^2\\
&-2(-(F+H)^2+G(G-4)+E^2+4)x'^2z'^2\\
&-2(-(F-G)^2+H(H+4)+E^2+4)z'^2t'^2\\
&-2(-(G-H)^2+F(F+4)+E^2+4)y'^2z'^2\\
&-2(-(F-H)^2+G(G+4)+E^2+4)y'^2t'^2\\
&+8E^2x'y'z't' = 0.
\end{aligned} \quad (12)$$

We set $t' = 1$, and compute $x' = \sqrt{\alpha/\delta}$ and $y' = \sqrt{\beta/\delta}$. Then, can we solve (12) for z', since this equation has become linear in z': apart from $8E^2x'y'z't'$, all other terms are known, as they only involve the square of z'.

Knowing $(x' : y' : z' : t')$, we recover $(x : y : z : t)$ in the same manner: we set $t = 1$, and compute x and y by square root extractions. Then, we recover z by solving (11), which has become linear.

To summarize, a halving in $\mathcal{K}$ requires to take four square roots, and to do a few multiplications or divisions; by what was said above, we can actually predict that at most one of the square roots will require to extend the base field. Each square root is computed using the algorithm of Subsection 2.2; in the case where no root exists in the base field, we build a degree-2 extension, and correspondingly update the representation of the quantities we are using. In total, when $(\tilde{x} : \tilde{y} : \tilde{z} : \tilde{t})$ is defined over $\mathbb{F}_{p^e}$, the cost of halving is an expected $O(\mathsf{C}(e)\log(e) + \mathsf{M}(e)\log(p))$ operations in $\mathbb{F}_p$.

Summary. We briefly review the steps of the former algorithm.

Input: polynomials F, R, S, W, Z in $\mathbb{F}_p[T]$ that form the description of a divisor P as in (6); we write $\mathbb{F}_{p^e} = \mathbb{F}_p[T]/\langle F \rangle$

Output: Polynomials F', R', S', W', Z' that form the description of a divisor Q such that $[2]Q = P$.

(1) let $(\tilde{x} : \tilde{y} : \tilde{z} : \tilde{t})$ be the coordinates of the image of P on $\mathcal{K}$
(2) compute $(\alpha : \beta : \gamma : \delta) = (x'^2 : y'^2 : z'^2 : t'^2)$ by solving (9) and (10)
(3) using the algorithms of Subsection 2.2, compute an extension $\mathbb{F}_{p^e} \to \mathbb{F}_{p^{\tilde{e}}}$ that contains a square root x' of α/δ and a square root y' of β/δ; set $t' = 1$ and use (12) to find z'
(4) compute $(\alpha' : \beta' : \gamma' : \delta') = (x^2 : y^2 : z^2 : t^2)$ by solving (8)
(5) using the algorithms of Subsection 2.2, compute an extension $\mathbb{F}_{p^{\tilde{e}}} \to \mathbb{F}_{p^{e'}} = \mathbb{F}_p[T]/\langle F' \rangle$ that contains a square root x of α'/δ' and a square root y of β'/δ'; set $t = 1$ and use (11) to find z
(6) compute the preimage $Q = \langle X^2 + u_1 X + u_0, v_1 X + v_0 \rangle$ of $(x : y : z : t)$ on $\mathcal{J}$, let R', S', W', Z' be in $\mathbb{F}_p[T]$ such that $u_1 = R' \bmod F'$ and $\deg(R') < \deg(F')$, etc, and return F', R', S', W', Z'

Experimental results. Table 2 gives timings (in seconds) obtained for lifting 2^k-torsion for one curve defined over $\mathbb{F}_p$, with $p = 2^{127} - 1$. We see that it takes about 5 CPU days (on the same machine as in Subsection 3.5) to reach torsion of order $2^{17} = 131072$; this is typical of the general behavior.

The rows in the table give the time necessary to compute all required square roots, then the necessary Frobenius computations and search for (s_1, s_2), as explained in Subsection 4.1. Obviously, the bottleneck is the computation of square roots; doubling the degree of the base field over $\mathbb{F}_p$ induces (roughly) a four-fold increase in running time, consistent with the cost estimate (the dominant cost is $\mathsf{C}(e)\log(e)$, and $\mathsf{C}(e)$ is quadratic in e in the NTL implementation).

torsion	2^6	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}
deg e_k	2^5	2^6	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}
sqrt	0.3	0.9	2.6	8	27	93	322	1227	5396	20743	78089	350671
Frob	0.6	1.3	2.6	5	11	23	51	109	262	581	1188	3878
(s_1, s_2)	0.3	0.8	2.3	5	13	33	78	194	544	1540	6439	31791

Table 2. Timings in seconds for 2^k-torsion

4.4. Lifting the 3^k-torsion

We next describe the computation of 3^k-torsion divisors. As for 2^k-torsion, the issue we discuss here is how to perform division by 3 in the Jacobian.

On input $P \in \mathcal{J}$ defined over $\mathbb{F}_{p^e}$, our goal is to find $Q \in \mathcal{J}$ such that $P = [3]Q$; in view of Lemma 6, assuming that all points of $\mathcal{J}[3]$ are defined over $\mathbb{F}_{p^e}$ as well, we know that Q will be defined over $\mathbb{F}_{p^{e'}}$, with either $e' = e$ or $e' = 3e$. We will suppose that both P and Q have weight 2, writing

$$P = \langle X^2 + u_{1,P}X + u_{0,P},\ v_{1,P}X + v_{0,P} \rangle \quad \text{and} \quad Q = \langle X^2 + u_{1,Q}X + u_{0,Q},\ v_{1,Q}X + v_{0,Q} \rangle.$$

Then, finding Q amounts to solving a system of polynomial equations in $u_{0,Q}$, $u_{1,Q}$, $v_{0,Q}$, $v_{1,Q}$. Many solutions are available to achieve this goal; the one that did the best for our specific family of equations uses homotopy techniques, and is derived from Gaudry and Harley (2000).

Compared to our solution for 2^k-torsion, division by 3 requires much more work. In the former case, all the time was spent computing square roots, and it was straightforward to know which square roots to compute. Here, we end up doing root-finding in degree 3, but prior to this, a significant amount of time is spent handling multivariate equations.

Initial set of equations. Let $U_{0,P}, U_{1,P}, V_{0,P}, V_{1,P}, U_{0,Q}, U_{1,Q}, V_{0,Q}, V_{1,Q}$ be indeterminates, that represent the Mumford coordinates of P and Q. The equations expressing that

$$P \in \mathcal{J}, \quad Q \in \mathcal{J}, \quad P = [3]Q$$

yield polynomial equations in $U_{0,P}, U_{1,P}, V_{0,P}, V_{1,P}, U_{0,Q}, U_{1,Q}, V_{0,Q}, V_{1,Q}$. However, the expressions derived from $P = [3]Q$ are quite heavy; to obtain simpler ones, we replace the constraint $P = [3]Q$ by the equivalent one $P - Q = [2]Q$. Then, clearing denominators, we obtain

$$\mathbf{H} \left| \begin{array}{ll} h_{1,Q}(U_{0,Q}, U_{1,Q}, V_{0,Q}, V_{1,Q}) = 0 & H_1(U_{0,Q}, U_{1,Q}, V_{0,Q}, V_{1,Q}, U_{0,P}, U_{1,P}, V_{0,P}, V_{1,P}) = 0, \\ h_{2,Q}(U_{0,Q}, U_{1,Q}, V_{0,Q}, V_{1,Q}) = 0 & H_2(U_{0,Q}, U_{1,Q}, V_{0,Q}, V_{1,Q}, U_{0,P}, U_{1,P}, V_{0,P}, V_{1,P}) = 0, \\ h_{1,P}(U_{0,P}, U_{1,P}, V_{0,P}, V_{1,P}) = 0 & H_3(U_{0,Q}, U_{1,Q}, V_{0,Q}, V_{1,Q}, U_{0,P}, U_{1,P}, V_{0,P}, V_{1,P}) = 0, \\ h_{2,P}(U_{0,P}, U_{1,P}, V_{0,P}, V_{1,P}) = 0 & H_4(U_{0,Q}, U_{1,Q}, V_{0,Q}, V_{1,Q}, U_{0,P}, U_{1,P}, V_{0,P}, V_{1,P}) = 0, \\ & \Lambda(U_{0,Q}, U_{1,Q}, V_{0,Q}, V_{1,Q}, U_{0,P}, U_{1,P}, V_{0,P}, V_{1,P}) \neq 0 \end{array} \right.$$

where the polynomials $(h_{1,P}, h_{2,P})$ and $(h_{1,Q}, h_{2,Q})$ express that P and Q belong to the Jacobian, (H_1, H_2, H_3, H_4) express $P - Q = [2]Q$, by equating abscissa and ordinates of both sides, and Λ is the product of all denominators appearing in the addition formulae. In all rigor, one should also consider the degenerate cases where $\Lambda = 0$; however, this was never needed in our experiments.

To highlight the structure of the solution set, we use the action of the 3-torsion, following an idea introduced by Gaudry and Schost (2004a) for 2^k-torsion. As an abstract group, $\mathcal{J}[3]$ is isomorphic to $(\mathbb{Z}/3\mathbb{Z})^4$. Consider subgroups

$$G_0 = \{0\} \subset G_1 \simeq (\mathbb{Z}/3\mathbb{Z}) \subset G_2 \simeq (\mathbb{Z}/3\mathbb{Z})^2 \subset G_3 \simeq (\mathbb{Z}/3\mathbb{Z})^3 \subset \mathcal{J}[3] \simeq (\mathbb{Z}/3\mathbb{Z})^4.$$

In what follows, we let q be such that all points of $\mathcal{J}[3]$ are defined over $\mathbb{F}_q$. Then, to G in $\mathcal{J}[3]$, we associate the rational function $U_{0,Q}^G \in \mathbb{F}_q(U_{0,Q}, U_{1,Q}, V_{0,Q}, V_{1,Q})$ that denotes the U_0-coordinate of $Q + G$.

Then, to each subgroup G_i, we associate $\Sigma_i = \sum_{G \in G_i} U_{0,Q}^G$, so that $\Sigma_0 = U_{0,Q}$: these are orbit-sums under the actions of G_0, G_1, G_2, G_3. We introduce new variables S_3, S_2, S_1, and add to $\mathbf{H}$ the polynomials obtained by taking the numerators of the rational functions $S_i - \Sigma_i(U_{0,Q}, U_{1,Q}, V_{0,Q}, V_{1,Q})$, for $i = 1, 2, 3$, and multiply Λ by the denominators of these rational functions. Remark that now, $\mathbf{H}$ is defined with coefficients in $\mathbb{F}_q$.

A triangular Gröbner basis. The system $\mathbf{H}$ is of dimension 2, so it is natural to consider the system $\mathbf{H}$ over the base field $\mathbb{F}_q(U_{0,P}, U_{1,P})$; to take into account the inequation $\Lambda \neq 0$, we add $1 - N\Lambda$ to $\mathbf{H}$, where N is a new variable. Then, we observe experimentally that the system $\mathbf{H}$ is zero-dimensional over $\mathbb{F}_q(U_{0,P}, U_{1,P})$, and that its Gröbner basis for the lexicographic order $N > V_{1,Q} > V_{0,Q} > U_{1,Q} > U_{0,Q} > S_1 > S_2 > S_3 > V_{1,P} > V_{0,P}$

has the following triangular form:

$$\mathbf{T} \left| \begin{array}{l} \mathbf{N} - R(V_{0,P}, S_3, S_2, S_1, U_{0,Q}), \\ \mathbf{V}_{1,Q} - L_1(V_{0,P}, S_3, S_2, S_1, U_{0,Q}), \\ \mathbf{V}_{0,Q} - L_0(V_{0,P}, S_3, S_2, S_1, U_{0,Q}), \\ \mathbf{U}_{1,Q} - M_1(V_{0,P}, S_3, S_2, S_1, U_{0,Q}), \\ T_0(V_{0,P}, S_3, S_2, S_1, \mathbf{U_{0,Q}}) \\ T_1(V_{0,P}, S_3, S_2, \mathbf{S_1}) \\ T_2(V_{0,P}, S_3, \mathbf{S_2}) \\ T_3(V_{0,P}, \mathbf{S_3}), \\ \mathbf{V}_{1,P} - N_1(V_{0,P}), \\ N_0(\mathbf{V}_{0,P}), \end{array} \right. \tag{13}$$

where the leading variables are written in bold; note that all coefficients are in $\mathbb{F}_q(U_{0,P}, U_{1,P})$. The polynomial N_0 is biquadratic in its main variable $V_{0,P}$ and the polynomials $T_0, \ldots, T_3$ have degree 3 in their main variables; thus, $\mathbf{H}$ has 324 solutions. The geometric interpretation of this number of solutions is that for each pair $(U_{0,P}, U_{1,P})$, there are 4 distinct ways to complete it to a valid Mumford representation with $(V_{0,P}, V_{1,P})$, as seen in the fact that N_0 has degree 4 in its main variable $V_{0,P}$. Then, each of these points has 81 preimages by division by 3. The benefit of introducing S_1, S_2, S_3 appears here: they allow us to decompose a degree-81 extension into 4 extensions of degree 3.

For $(u_{0,P}, u_{1,P})$ in $\mathbb{F}_{p^e}$ (where e is such that q divides p^e), we write $\mathbf{H}_{u_{0,P},u_{1,P}}$ to denote the system $\mathbf{H}$ where $(U_{0,P}, U_{1,P})$ have been specialized at $(u_{0,P}, u_{1,P})$. Similarly, we denote by $\mathbf{T}_{u_{0,P},u_{1,P}}$ the specialization of $\mathbf{T}$ at $(U_{0,P}, U_{1,P}) = (u_{0,P}, u_{1,P})$, assuming no denominator vanishes.

We can now state the division-by-3 problem, and our solution, more precisely: given $(u_{0,P}, u_{1,P}, v_{0,P}, v_{1,P})$ in $\mathbb{F}_{p^e}$ (with the same constraint on e as above), we want to find an extension $\mathbb{F}_{p^{e'}}$ that contains the coordinates of one solution of $\mathbf{H}_{u_{0,P},u_{1,P}}$. This will be done by computing $\mathbf{T}_{u_{0,P},u_{1,P}}$; once this is done, since we know $V_{0,P}$ and $V_{1,P}$, it remains to find roots of T_3, T_2, T_1, T_0, in this order (each root-finding may involve extending the base field, and updating the representation of some elements of $\mathbb{F}_{p^e}$).

In terms of complexity, since the system $\mathbf{H}$ is fixed, computing $\mathbf{T}_{u_{0,P},u_{1,P}}$ takes a constant number of operations in the field of definition of $u_{0,P}, u_{1,P}$; with our previous notation, this is $O(\mathsf{M}(e)\log(e))$ operations in $\mathbb{F}_p$ – however, reducing the constant hidden in the big-O is crucial, and this is where we will direct our attention below. Using the results of Subsection 2.2, finding the extension of $\mathbb{F}_{p^e}$ that contains the solutions of $\mathbf{T}_{u_{0,P},u_{1,P}}$ then takes an expected $O(\mathsf{C}(e)\log(e) + \mathsf{M}(e)\log(p))$ operations in $\mathbb{F}_p$ (we will not discuss this part anymore here).

Homotopy techniques. There exist many ways to compute $\mathbf{T}_{u_{0,P},u_{1,P}}$: solving the system directly (using Gröbner bases, resultants, ...), computing once and for all the triangular set $\mathbf{T}$ over the rational function field $\mathbb{F}_q(U_{0,P}, U_{1,P})$, etc. As remarked before, since $\mathbf{H}$ is fixed, the cost of all these solutions is the same as far as we stick to the big-O notation

(the differences are in the hidden constant). The solution we present here is the one that did best in practice.

We start by constructing $\mathbb{F}_q$ such that the points of $\mathcal{J}[3]$ are defined over $\mathbb{F}_q$. Then, we find a starting point $(u_{0,P'}, u_{1,P'})$, such that all 324 solutions of the system $\mathbf{H}_{u_{0,P'},u_{1,P'}}$ are known, and are in $\mathbb{F}_q$. This is done by constructing Q' and Q'' such that $P' = [3]Q'$ and $P'' = [3]Q''$ have the same U-polynomials and are non-opposite, and by letting $\mathcal{J}[3]$ act on Q', $-Q'$, Q'' and $-Q''$, giving us generically the requested 324 solutions. In this setting $u_{0,P'}$ and $u_{1,P'}$ are the coefficients of the common U-polynomial of P' and P''.

To obtain Q' and Q'', we start from a random divisor D of weight 1, and let $E = [3]D$; next, we find divisors D' and D'' defined over $\mathbb{F}_p$ such that $F' = E + [3]D'$ and $F'' = E + [3]D''$ have weight 1 (this is simply done by writing down and solving the equations for the Mumford coordinates for such D' and D''; if there is no solution in $\mathbb{F}_q$, we choose another D). Then, we take $Q' = -D' + D''$ and $Q'' = [2]D + D' + D''$; one checks that $[3]Q' = -F' + F''$ and $[3]Q'' = F' + F''$, as needed.

In what follows, we assume that e is such that all of P, P', P'', Q', Q'' and all points of $\mathcal{J}[3]$ are defined over $\mathbb{F}_{p^e}$. Starting from $\mathbf{H}_{u_{0,P'},u_{1,P'}}$, we will use a homotopy continuation to solve $\mathbf{H}_{u_{0,P},u_{1,P}}$. Let t be a new variable and let

$$\tau_0 = tu_{0,P} + (1-t)u_{0,P'}, \quad \tau_1 = tu_{1,P} + (1-t)u_{1,P'}.$$

We will consider the system $\mathbf{H}_{\tau_0,\tau_1}$ and the associated triangular set $\mathbf{T}_{\tau_0,\tau_1}$; both of them have coefficients in the rational function field $\mathbb{F}_{p^e}(t)$. Specializing t at 0, we obtain the system $\mathbf{H}_{u_{0,P'},u_{1,P'}}$, whose solutions are known; specializing t at 1, we get the system $\mathbf{H}_{u_{0,P},u_{1,P}}$ that we want to solve.

We compute $\mathbf{T}_{\tau_0,\tau_1}$ using Newton iteration. Let $\mathbf{H}'$ be the square subsystem

$$\mathbf{H}' = (h_{1,P}, h_{2,P}, H_1, H_2, H_3, H_4)$$

extracted from $\mathbf{H}$, and let us assume that the Jacobian determinant of $\mathbf{H}'$ vanishes nowhere on the known solutions of $\mathbf{H}_{u_{0,P'},u_{1,P'}}$ – experimentally, we observe that this is the case for a generic choice of Q' and Q''. Using Newton iteration, we lift all the roots of $\mathbf{H}'_{u_{0,P'},u_{1,P'}}$ to 324 roots of $\mathbf{H}'_{\tau_0,\tau_1}$ with coordinates in $\mathbb{F}_{p^e}[[t]]$. Note that these roots are actually the roots of the whole system $\mathbf{H}_{\tau_0,\tau_1}$, by the uniqueness property of Newton iteration.

From these roots, one can recover $\mathbf{T}_{\tau_0,\tau_1}$ using interpolation techniques: we know the values of the indeterminates $V_{0,P}, V_{1,P}, U_{0,Q}, \ldots, V_{1,Q}$, which is enough to recover those of S_1, S_2, S_3 (since they are rational functions of the former). Then, $\mathbf{T}_{\tau_0,\tau_1}$ defines the vanishing ideal of these points, and is obtained using interpolation formulae as in Dahan and Schost (2004).

Since we know the power series expansions of the roots of $\mathbf{T}_{\tau_0,\tau_1}$, the interpolation is conducted with power series coefficients. As a result, we do not obtain $\mathbf{T}_{\tau_0,\tau_1}$ directly, but $\mathbf{T}_{\tau_0,\tau_1}$ with all coefficients expanded in $\mathbb{F}_{p^e}[[t]]$. We recover the rational functions in $\mathbb{F}_{p^e}(t)$ by means of rational function reconstruction, and eventually set $t = 1$ to get $\mathbf{T}_{u_{0,P},u_{1,P}}$.

Improving the lifting. We mention here improvements over a naive lifting algorithm, in decreasing order of importance. The most important saving comes from using the action of the 3-torsion: once a solution (P, Q) is known, then the 162 pairs $(P, Q + G)$ and $(-P, -Q + G)$, for G in $\mathcal{J}[3]$, are solutions as well. Thus, we need only to lift two solutions, to recover all 324 of them by conjugations.

Secondly, we use the fact that the equations in $\mathbf{H}'$ can be evaluated using a small number of operations to speed up the lifting. Indeed, almost all the time in Newton iteration is spent evaluating the system and its Jacobian matrix on the current approximate solution. In expanded form, the polynomials in $\mathbf{H}'$ total more than 80,000 monomials; instead, we use a straight-line program derived from the group law formulae, that performs only 60 multiplications (about 180 for the Jacobian matrix).

Next, the interpolation formulae we use are not the straightforward ones, as we do not interpolate $\mathbf{T}_{\tau_0,\tau_1}$ itself. For the first polynomials N_0 and N_1, nothing changes. However, starting from T_3, we slightly modify our objective: instead of interpolating T_3, we work with $(\partial N_0/\partial V_{0,P})T_3$; similar modifications apply to the other polynomials. The net effect of this transformation is to reduce the degree in t of the coefficients, and thus the required precision for our power series roots, from several thousands to 81 (this was determined experimentally); this is a general phenomenon, detailed in Dahan and Schost (2004).

A last improvement comes from exploiting the structure of the system $\mathbf{H}'$, that involves the 6 variables $V_{0,P}, V_{1,P}, U_{0,Q}, U_{1,Q}, V_{0,Q}, V_{1,Q}$. Since it admits the square subsystem $(h_{1,P}, h_{2,P})$ which depends only on $V_{0,P}$ and $V_{1,P}$, we can lift these two coordinates first, then deal with the 4 remaining unknowns $U_{0,Q}, \ldots, V_{1,Q}$ using the equations $H_1, \ldots, H_4$ (so we split our 6×6 problem into a 2×2 one and a 4×4 one).

Summary. We briefly review the steps of the former algorithm.

Input:

- polynomials F, R, S, W, Z in $\mathbb{F}_p[T]$ that form the description of a divisor P as in (6); we write $\mathbb{F}_{p^e} = \mathbb{F}_p[T]/\langle F\rangle$ and $P = \langle X^2 + u_{1,P}X + u_{0,P}, v_{1,P}X + v_{0,P}\rangle$, with $u_{1,P} = R \bmod F$, etc
- the coordinates of all elements of $\mathcal{J}[3]$ (we assume that all these coordinates belong to $\mathbb{F}_{p^e}$)

Output: Polynomials F', R', S', W', Z' that form the description of a divisor Q such that $[3]Q = P$.

(1) find $P', Q'P'', Q''$ in $\mathcal{J}(\mathbb{F}_{p^e})$ such that $P' = [3]Q'$ and $P'' = [3]Q''$ have the same U-polynomials; let $u_{0,P'}$, etc, be their Mumford coordinates
(2) let $\tau_0 = tu_{0,P} + (1-t)u_{0,P'}$ and $\tau_1 = tu_{1,P} + (1-t)u_{1,P'}$
(3) using Newton iteration, compute vectors

$$W' = (\nu'_0, \nu'_1, \mu'_0, \mu'_1, \eta'_0, \eta'_1), \quad W'' = (\nu''_0, \nu''_1, \mu''_0, \mu''_1, \eta''_0, \eta''_1)$$

in $\mathbb{F}_{p^e}[t]^6$ such that $\mathbf{H}'_{\tau_0,\tau_1}(W') = 0 \bmod t^{81}$, $\mathbf{H}'_{\tau_0,\tau_1}(W'') = 0 \bmod t^{81}$ and such that

$$W' \bmod t = (v_{0,P'}, v_{1,P'}, u_{0,Q'}, u_{1,Q'}, v_{0,Q'}, v_{1,Q'})$$

and

$$W'' \bmod t = (v_{0,P''}, v_{1,P''}, u_{0,Q''}, u_{1,Q''}, v_{0,Q''}, v_{1,Q''}).$$

(4) let $\pi', \pi'', \rho', \rho''$ be the "divisors" with Mumford representations

$$\pi' = \langle X^2 + \tau_1 X + \tau_0, \nu'_1 X + \nu'_0 \rangle, \quad \rho' = \langle X^2 + \mu'_1 X + \mu'_0, \eta'_1 X + \eta'_0 \rangle$$

and

$$\pi'' = \langle X^2 + \tau_1 X + \tau_0, \nu''_1 X + \nu''_0 \rangle, \quad \rho'' = \langle X^2 + \mu''_1 X + \mu''_0, \eta''_1 X + \eta''_0 \rangle,$$

and let $(\rho'_i)_{1 \le i \le 162}$ and $(\rho''_i)_{1 \le i \le 162}$ be the divisors obtained by adding all elements in $\mathcal{J}[3]$ to $\rho', -\rho'$, and $\rho'', -\rho''$; all computations are done modulo t^{81}

(5) interpolate $\mathbf{T}_{\tau_0,\tau_1}$ from its roots, which are the coefficients of the divisors computed above; all computations are done modulo t^{81}

(6) do rational reconstruction on all coefficients of $\mathbf{T}_{\tau_0,\tau_1}$, and compute $\mathbf{T}_{u_{0,P},u_{1,P}}$ by letting $t = 1$

(7) using the algorithms of Subsection 2.2, compute an extension $\mathbb{F}_{p^e} \to \mathbb{F}_{p^{e'}} = \mathbb{F}_p[T]/\langle F' \rangle$ that contains a root $(v_{0,P}, v_{1,P}, \sigma_3, \sigma_2, \sigma_1, u_{0,Q}, u_{1,Q}, v_{0,Q}, v_{1,Q})$ of the system $\mathbf{T}_{u_{0,P},u_{1,P}}$, and return the polynomials F', R', S', W', Z', with $u_{1,Q} = R' \bmod F'$ and $\deg(R') < \deg(F')$, etc

Experimental results. Table 3 gives timings (in seconds) obtained for lifting 3^k-torsion for one curve defined over $\mathbb{F}_p$, with $p = 2^{127} - 1$. The timings comply rather closely with theoretical predictions. Indeed, from torsion index 3^k to 3^{k+1}, the degree e_k is multiplied by 3; the time for root-finding is (roughly) multiplied by 9 or 10 (revealing a quadratic running time), whereas the time spent in the other operations grows essentially linearly. To summarize, this table represents about 1 CPU day; timings from 1 to 2 CPU days to reach torsion index 729 or 2187 are typical (depending on the degree in which we find the initial torsion divisor P_1).

index 3^k	3^2	3^3	3^4	3^5	3^6
degree e_k	$10 \cdot 3$	$10 \cdot 3^2$	$10 \cdot 3^3$	$10 \cdot 3^4$	$10 \cdot 3^5$
lifting	18	84	308	1356	4325
action of $\mathcal{J}[3]$	37	220	678	3325	11733
interpolation	66	334	1065	4629	14977
root-finding	4	34	339	2683	31898
Frobenius	0.6	2.3	9	21	95
finding (s_1, s_2)	0.2	1.2	9	31	160

Table 3. Timings in seconds for 3^k-torsion

4.5. Lifting the 5^k and 7^k-torsion

We conclude this section with the description of the computation of 5^k- and 7^k-torsion divisors: as before, our actual question is how to perform division by 5 or 7 in the Jacobian. For conciseness, we give details here for division by 5, and mention in the end the modifications for division by 7.

On input $P \in \mathcal{J}$ defined over $\mathbb{F}_{p^e}$, our goal is thus to find $Q \in \mathcal{J}$ such that $P = [5]Q$; in view of Lemma 6, if we assume that all points of $\mathcal{J}[5]$ are defined over $\mathbb{F}_{p^e}$ as well, we

know that Q will be defined over $\mathbb{F}_{p^{e'}}$, with either $e' = e$ or $e' = 5e$. As before, we will suppose that Q has weight 2; then, finding it amounts to solving a system of polynomial equations in its Mumford coordinates.

We used a more direct approach than in the other cases, based on resultant computations. The strategy used to lift 3^k-torsion would be applicable here as well, but becomes inferior (and of course, the explicit formulae using square roots are specific to 2^k-torsion).

Input and output. The equation $P = [5]Q$ is rewritten as $P - [2]Q = [3]Q$, so as to balance the degrees of both sides. Letting (U_0, U_1, V_0, V_1) be indeterminates that represent the coordinates of Q, and taking coordinates in the former relation, we obtain the system (with coefficients in $\mathbb{F}_{p^e}$)

$$\mathbf{K} \left| \begin{array}{ll} h_1(U_0, U_1, V_0, V_1) = 0 & K_1(U_0, U_1, V_0, V_1) = 0, \\ h_2(U_0, U_1, V_0, V_1) = 0 & K_2(U_0, U_1, V_0, V_1) = 0, \\ & K_3(U_0, U_1, V_0, V_1) = 0, \\ & K_4(U_0, U_1, V_0, V_1) = 0, \\ & \Lambda(U_0, U_1, V_0, V_1) \neq 0, \end{array} \right. \tag{14}$$

where Λ is the product of denominators that arise when applying the group law operations. The equations (h_1, h_2) encode the fact that Q is in $\mathcal{J}$; they are obtained as the coefficients of $((V_1X + V_0)^2 - f(X)) \bmod (X^2 + U_1X + U_0)$. Given these equations, we will show here how to compute a representation of the solutions of the form

$$\left| \begin{array}{rcl} V_1 & = & D(U_1) \\ V_0 & = & C(U_1) \\ U_0 & = & B(U_1) \\ A(U_1) & = & 0, \end{array} \right. \tag{15}$$

where all polynomials have coefficients in $\mathbb{F}_{p^e}$.

The existence of such a representation is not guaranteed. For any divisor P, there exist $5^4 = 625$ divisors Q such that $P = [5]Q$; however, some of them may have weight 1, or cancel the polynomial Λ, and thus may not be solutions of $\mathbf{K}$. Even if there are 625 solutions, they may not admit a description of the given shape.

We do not take such degenerate cases into account, and consider only the generic case where $\mathbf{K}$ has 625 solutions, and admits a description as claimed (then, A has degree 625); if we are not in this favorable situation, we abort the computation.

The core of this subsection explains how to compute the polynomials A, B, C, D. Once this is done, it remains to find a root of A in an extension of $\mathbb{F}_{p^e}$: as said above, we know that we will find such a root in $\mathbb{F}_{p^{e'}}$, with either $e' = e$ or $e' = 5e$; then, it suffices to rewrite B, C, D as polynomials over $\mathbb{F}_{p^{e'}}$ and evaluate them at the said root. All this is done using the algorithm of Subsection 2.2, and will not be explained anymore here. We simply point out that it would be possible to use the action of $\mathcal{J}[5]$ to replace the root-finding in degree 625 by 4 root-findings in degree 5, as we did for 3-torsion; however, root-finding was not a bottleneck, so we did not implement this idea.

In terms of complexity, the cost is theoretically dominated by the root-finding. Indeed, computing (A, B, C, D) takes a constant number of operations in $\mathbb{F}_{p^e}$, for a total of $O(\mathsf{M}(e)\log(e))$ operations in $\mathbb{F}_p$; as mentioned in Subsection 2.2, root-finding in fixed degree over $\mathbb{F}_{p^e}$ takes an expected $O(\mathsf{C}(e)\log(e)+\mathsf{M}(e)\log(p))$ operations in $\mathbb{F}_p$. However, we will see that theory and practice did not always agree in our experiments.

Solving the system. Our strategy to compute A, B, C, D is to first eliminate (V_0, V_1) from $\mathbf{K}$, so as to be left with a bivariate system in (U_0, U_1); we solve the latter using bivariate resultant techniques.

We eliminate (V_0, V_1) by solving the equations $h_1 = h_2 = 0$, obtaining

$$V_1 - E_3V_0^3 - E_1V_0, \quad V_0^4 + F_2V_0^2 + F_0 = 0, \tag{16}$$

where E_1, E_3 and F_0, F_2 are simple rational functions of (U_0, U_1). Since the equations h_1, h_2 are quite simple, this is done by computing a Gröbner basis of (h_1, h_2) for the lexicographic order $V_1 > V_0$ in $\mathbb{F}_{p^e}(U_0, U_1)[V_0, V_1]$ (the coefficients E_1, E_3, F_0, F_2 can actually be precomputed and stored).

Let $\varphi_1, \varphi_2, \varphi_3$ be obtained by reducing K_1, K_2, K_3 modulo the polynomials in (16), so that all φ_i are in $\mathbb{F}_{p^e}(U_0, U_1)[V_0]$ and have degree at most 3 in V_0. For $i = 1, 2, 3$, we define further

$$\gamma_i = \mathrm{res}(\varphi_i, V_0^4 + F_2V_0^2 + F_0, V_0)$$

which is thus in $\mathbb{F}_{p^e}(U_0, U_1)$; we then let $G_i \in \mathbb{F}_{p^e}[U_0, U_1]$ be obtained by a cleaning process from γ_i: we clear denominators and remove predictable parasites (this process is described in more detail later on). Then, we compute the polynomials A, B, C, D of (15) as follows:

- The polynomial $A(U_1)$ is given by

$$A_{1,2} = \mathrm{res}(G_1, G_2, U_0), \quad A_{1,3} = \mathrm{res}(G_1, G_3, U_0), \quad A = \gcd(A_{1,2}, A_{1,3}).$$

- The polynomial $B(U_1)$ is computed by

$$B_1U_0 + B_0 = \mathrm{sres}(G_1, G_2, U_0), \quad B = -B_0/B_1 \bmod A.$$

- To compute $C(U_1)$, we let $\psi_1 = \varphi_1(B, U_1, V_0) \bmod A$; this polynomial belongs to $\mathbb{F}_{p^e}[U_1, V_0]$ and has degree 3 in V_0. We compute its GCD with $V_0^4 + F_2(B, U_1)V_0^2 + F_0(B, U_1)$ modulo A, using two steps of the Euclidean GCD algorithm. This GCD has the form $C_1(U_1)V_0 + C_0(U_1)$, and we get $C = -C_0/C_1 \bmod A$.
- Finally, D is given by $D = E_3(B, U_1)C^3 + E_1(B, U_1)C \bmod A$.

Provided all steps are well-defined, and provided the parasite factors we remove indeed describe parasite solutions, the specialization properties of resultants imply that the solutions described by the polynomials A, B, C, D are indeed solutions of the sub-system $h_1 = h_2 = K_1 = K_2 = K_3 = 0$. Experimentally, we observed that we obtain in this way all solutions of the whole system $\mathbf{K}$.

Implementation details. We start by explaining how we compute G_1, G_2, G_3. First, we define some predictable parasite factors p_1, p_2, p_3, p_4 in $\mathbb{F}_{p^e}[U_0, U_1, V_0, V_1]$: p_1 and p_2 are given by

$$\begin{aligned} p_1 = {} & V_1^3 + V_1U_1^3 - f_4V_1U_1^2 - 4V_1U_1U_0 + f_3V_1U_1 + 2f_4V_1U_0 - f_2V_1 \\ & +3V_0U_1^2 - 2f_4V_0U_1 - 2V_0U_0 + f_3V_0; \\ p_2 = {} & V_1^2U_0 - V_1V_0U_1 + V_0^2, \end{aligned}$$

where the f_i are the coefficients of the polynomial f defining $\mathcal{C}$. Two additional parasites p_3, p_4 are obtained as denominators arising when computing $[3]Q$ and $P - [2]Q$; they are too large to be printed here. These parasites were obtained by computing and factoring resultants (over small base fields, to make computations easy) and either reading them off (in the case of p_1, p_2) or matching them with denominators appearing in the group law.

Let P_1, P_2, P_3, P_4 be obtained by reducing these equations modulo the polynomials in (16). For $i = 1, 2, 3, 4$, we define

$$\pi_i = \mathrm{res}(P_i, V_0^4 + F_2 V_0^2 + F_0, V_0)$$

and we set

$$g_1 = \frac{(4U_0 - U_1^2)^{36}\gamma_1}{\pi_1^{16}\pi_2^6}, \quad g_2 = \frac{(4U_0 - U_1^2)^{36}\gamma_2}{\pi_1^{16}\pi_2^6}, \quad g_3 = \frac{(4U_0 - U_1^2)^{49}\gamma_3}{\pi_1^{43}\pi_2^{14}\pi_3\pi_4^3}. \tag{17}$$

The exponents $36, 16, \ldots$ have been found experimentally to rid $\gamma_1, \gamma_2, \gamma_3$ of predictable parasite factors, and clear denominators, so that g_1, g_2, g_3 are in $\mathbb{F}_{p^e}[U_0, U_1]$. These are almost the polynomials we want: G_1, G_2, G_3 are obtained by cleaning some further parasite factors (that we were not able to express as simply as $\pi_1, \pi_2, \pi_3, \pi_4$), by keeping only the degree-1 part in the squarefree decomposition of g_1, g_2, g_3.

We compute G_1, G_2, G_3 using evaluation and interpolation techniques, by computing their values for sufficiently many values (u_0, u_1) of (U_0, U_1) and interpolating them; as before, we use interpolation at a geometric progression. For any given value (u_0, u_1), the polynomials $\varphi_1, \varphi_2, \varphi_3$ are computed using a straight-line program that computes the coordinates of $[3]Q$ and $P - [2]Q$, and equates them; all operations in this straight-line program are done modulo the polynomials in (16) (where (U_0, U_1) are specialized at (u_0, u_1)). The parasites are then cleaned (before interpolation); the squarefree decompositions are computed after interpolating u_1, and before interpolating u_0.

Once G_1, G_2, G_3 are known, $A_{1,2}, A_{1,3}$ and B_0, B_1 are computed using the evaluation and interpolation techniques described in Subsection 2.1.

This concludes our explanations for division by 5. In the case of division by 7, we were not able to predict such simple parasite factors; as a result, we have to interpolate polynomials of larger degrees, before taking squarefree parts. Table 4 gives information on the degrees of the polynomials we compute using this approach: remark in particular that the degrees of $A_{1,2}$ and $A_{1,3}$ are much larger than that of their GCD A.

index	$\deg((G_1, G_2, G_3), U_0)$	$\deg((G_1, G_2, G_3), U_1)$	$\deg(A_{1,2})$	$\deg(A_{1,3})$	$\deg(A)$
5	$(100, 100, 168)$	$(98, 100, 164)$	10000	16800	625
7	$(196, 196, 296)$	$(194, 196, 292)$	38416	58016	2401

Table 4. Degrees appearing in the process of division by 5 or 7

Summary. We briefly review the steps of the division-by-5 algorithm described above (the case of division by 7 is similar, as we explained). Hereafter, U_0, U_1, V_0, V_1 are indeterminates that represent the coordinates of the unknown divisor Q.

Input: Polynomials F, R, S, W, Z in $\mathbb{F}_p[T]$ that form the description of a divisor P as in (6); we write $\mathbb{F}_{p^e} = \mathbb{F}_p[T]/\langle F \rangle$

Output: Polynomials F', R', S', W', Z' that form the description of a divisor Q such that $[5]Q = P$.

(1) compute the equations h_1, h_2, K_1, K_2, K_3 as in (14); they have coefficients in $\mathbb{F}_{p^e}$
(2) compute the Gröbner basis $\Gamma = (V_1 - E_3V_0^3 - E_1V_0, V_0^4 + F_2V_0^2 + F_0)$ of (h_1, h_2) in $\mathbb{F}_{p^e}(U_0, U_1)[V_0, V_1]$, for the lexicographic order $V_1 > V_0$.
(3) for $i = 1, 2, 3$, compute the remainder $\varphi_i \in \mathbb{F}_{p^e}(U_0, U_1)[V_0]$ of K_i modulo Γ and $\gamma_i = \mathrm{res}(\varphi_i, V_0^4 + F_2V_0^2 + F_0, V_0)$
(4) for $i = 1, 2, 3, 4$, compute the parasite p_i, the remainder P_i of p_i modulo Γ, and $\pi_i = \mathrm{res}(P_i, V_0^4 + F_2V_0^2 + F_0, V_0)$
(5) for $i = 1, 2, 3$, compute g_i using (17) and compute G_i by keeping only the degree-1 part of the squarefree decomposition of g_i
(6) compute $A_{1,2} = \mathrm{res}(G_1, G_2, U_0)$, $A_{1,3} = \mathrm{res}(G_1, G_3, U_0)$ and $A = \gcd(A_{1,2}, A_{1,3})$
(7) compute $B_1U_0 + B_0 = \mathrm{sres}(G_1, G_2, U_0)$ and $B = -B_0/B_1 \bmod A$
(8) compute $\psi_1 = \varphi_1(B, U_1, V_0) \bmod A$, $C_0 + C_1V_0 = \gcd(\psi_1, V_0^4 + F_2(B, U_1)V_0^2 + F_0(B, U_1))$ over $\mathbb{F}_{p^e}[U_1]/\langle A\rangle$ and $C = -C_0/C_1 \bmod A$
(9) compute $D = E_3(B, U_1)C^3 + E_1(B, U_1)C \bmod A$
(10) using the algorithms of Subsection 2.2, compute an extension $\mathbb{F}_{p^e} \to \mathbb{F}_{p^{e'}} = \mathbb{F}_p[T]/\langle F'\rangle$ that contains a root u_1 of A and let $u_0 = B(u_1)$, $v_0 = C(u_1)$, $v_1 = D(u_1)$
(11) Let R', S', W', Z' in $\mathbb{F}_p[T]$ be such that $u_1 = R' \bmod F'$ and $\deg(R') < \deg(F')$, etc, and return F', R', S', W', Z'

Experimental results. In Tables 5 and 6, we give timings (in seconds) for division by 5 and by 7, for curves defined over $\mathbb{F}_p$, with $p = 2^{127} - 1$, as before. In the degrees we managed to reach, root-finding is not yet the bottleneck (although it becomes increasingly important). Lifting 7^k-torsion is much harder than lifting 5^k-torsion: the degree of the initial field extension is usually higher, and we have many more resultants to compute; practically, it usually did not make sense to try to reach index $7^3 = 343$.

index 5^k	5^2	5^3	5^4
degree e_k	$3 \cdot 5$	$3 \cdot 5^2$	$3 \cdot 5^3$
G_1, G_2, G_3	445	2993	35908
$A_{1,2}, A_{1,3}$	1732	17957	311993
A, B, C, D	34	249	1578
root-finding	53	2065	87746
Frobenius	0.1	1.7	8.1
finding (s_1, s_2)	0.1	0.5	9

Table 5. Timings in seconds for 5^k-torsion

5. Computation of a cryptographically secure curve

We conclude this paper by the description of large-scale computations that were conducted in order to discover a curve of genus 2, with desirable security and efficiency properties. Our purpose was to find a twist-secure curve (we define this precisely below).

index 7^k	7^2
degree e_k	$8 \cdot 7$
G_1, G_2, G_3	7115
$A_{1,2}, A_{1,3}$	113890
A, B, C, D	662
root-finding	10630
Frobenius	1.2
finding (s_1, s_2)	0.3

Table 6. Timings in seconds for 7^k-torsion

A crude simulation (assuming that the coefficients s_1 and s_2 have a uniform distribution in the admissible domain) showed that using an early abort strategy, one may hope to find such a curve after completing the point-counting for about 2000 curves, for an estimated running time of about 2,000,000 CPU hours. As it turns out, we found such a curve, in about half the time.

Security and efficiency constraints. Our first motivation for designing and implementing point counting algorithms is public key cryptography: we want to find a curve of genus 2 over a prime field that is suitable for building a public key cryptosystem. For security reasons, the order of the Jacobian of the curve must be prime or be a small multiple of a prime, and this prime must be large enough, so that the best known approach for solving the discrete logarithm problem in this group takes an unrealistic time.

With current technology, a security level of 2^{128} is considered as appropriate for many applications, meaning that the best known attack takes about that number of elementary operations (this last notion is vague: it can be an operation in the group, or one application of a hash function, or one application of the AES block cipher). To get a good compromise between fitting the security level and efficiency considerations, we decided to search for a curve of genus 2 with the following properties:

- *Base field.* The base field is the prime field $\mathbb{F}_p$, with $p = 2^{127} - 1$. The Jacobian group has about 2^{254} elements, and if the curve is well chosen, the best known attack will require about 2^{127} operations on average. The prime p is a Mersenne prime, so that reduction modulo p can be made extremely fast compared to a generic prime of the same size.
- *Rationality conditions.* The fastest known group arithmetic for scalar multiplication in Jacobian of genus 2 curves works not with the Jacobian itself but with the Kummer surface (Gaudry, 2007). Some information is lost compared to the Jacobian, but in many cryptographic applications, this is enough.

  To get optimal efficiency, coordinates on the Kummer surface based on Theta functions have to be used, and they might require to work in an extension of the base field (which would imply an undesired additional cost). Therefore, not every curve will be suitable for us, but only those that satisfy some rationality conditions.

  In our search we will start from parameters of the Kummer surface, called Theta constants; to match the notation of Gaudry (2007), we will call them a^2, b^2, c^2, d^2 (so they are actually *squared* Theta constants). Choosing them in $\mathbb{F}_p$ enforces some

rationality conditions; a few additional conditions subsist (three quantities should be squares, to obtain a rational map to the Jacobian of a genus 2 curve), slightly restricting our search space. We refer to Section 7.3 of Gaudry (2007) for details.

- *Small coefficients.* The pseudo-group law on the Kummer surface involves some constants that depend only on the curve, and not on the points to be added.

  In Subsection 4.3, we recalled the doubling formula, that involves the constants y_0, z_0, t_0 and y'_0, z'_0, t'_0. These quantities also occur in the pseudo-group law; they are related to the Theta constants a, b, c, d of the Kummer surface by $y_0 = a/b$, $z_0 = a/c$, $t_0 = a/d$ (still with the notation of Gaudry (2007)).

  As it turns out, the fastest pseudo-group law formulas use the squares of these quantities (hence our choice of using a^2, b^2, c^2, d^2 as parameters). Having these constants small is enough to guarantee that all important quantities are small (say a few dozens). When this is the case, the implementation of the pseudo-group law on the Kummer surface can take advantage of this (e.g. replacing multiplications by large constants by a few additions), so our cryptosystem becomes faster. The potential gain is substantial and was first noticed by Bernstein (2006b).
- *Twist-security.* The Kummer surface is the same for the curve and its quadratic twist. This fact has implications in cryptography, because in some cases the computing device might believe it is working with the curve whereas the twist is involved. Having both the curve and its quadratic twist cryptographically secure will therefore save the computations that check that the device is not being fooled by an attacker (see Bernstein (2006a) for similar considerations for elliptic curve cryptosystems).

  In practice, this means that the Jacobians of the curve and of its quadratic twist should have a group order which is prime or a small multiple of a prime. The rationality conditions that we impose on the curves imply that the group orders are divisible by 16. Therefore we seek a curve for which both group orders can be written 16 times a prime; such curves will be called *twist-secure*.

We acknowledge the fact that the security level of our curves is not really 2^{128} but rather 2^{125}, since we lost one bit in the base field and 2 more bits due to the rationality conditions in the Kummer surface. We consider that this is not a real problem, since the unit in this security level estimate is anyway rather vague.

Description of available computing resources. Our computations were performed on clusters belonging to the SHARCNET grid computing facility. We got dedicated resources on two clusters with different features:

- *Whale:* 768 nodes each equipped with two dual-core Opteron 275 processors at 2.2 GHz, with 4 GB of central memory and a Gigabit ethernet network. This is a throughput cluster; hours on this cluster are relatively easy to obtain.
- *Bull:* 96 nodes each equipped with 4 mono-core Opteron 850 processors at 2.4 GHz, with 32 GB of central memory. The nodes are connected with a high end interconnect Quadrics Elan4. Due to the large amount of memory per node and the fast network, hours on this cluster are much harder to get.

Organization of the computation. It is difficult to predict the size of the coefficients that occur in the pseudo-group law in the Kummer surface from the hyperelliptic equation of the curve. Therefore, we start from the parameters a^2, b^2, c^2, d^2 of the Kummer surface, and we denote by $\mathcal{C}_{a^2,b^2,c^2,d^2}$ the corresponding curve.

We start by enumerating all possible 4-tuples (a^2, b^2, c^2, d^2) below a certain bound. There are numerous symmetries, and we keep only one 4-tuple per isomorphism class. We also eliminate the few 4-tuples that yield a Jacobian that is split, because this implies that the group order cannot contain a large prime.

We did not complete the point-counting for all these tuples: we used early abort techniques, to discard as early as possible non twist-secure curves. Thus, a first filter was quickly applied, in order to remove from the list the parameters corresponding to curves for which the group order of the Jacobian of the curve or of the twist is not 16 modulo 32, or is zero modulo 3, 5 or 7. Compared to a complete point counting, this is very fast, and was done before sending the curves to the clusters.

For the remaining curves, in order to take full advantage of the computing resources, we split a full point-counting job into various tasks, isolating the tasks that require more than the 4GB of memory of a node of the Whale cluster. Tasks are separated in 3 classes:

- *Tiny memory.* For ℓ prime, the main part of the computation is the computation of (sub)resultants of polynomials of the form $(\mathfrak{E}_1(U_0, u_1), \mathfrak{E}_2(U_0, u_1))$, for various values of u_1, as described in Subsection 3.2. Each computation is very light (these polynomials have degree about ℓ^2, which is a few thousands), but we need about ℓ^4 of them. The computation was done in a distributed fashion, split across several nodes into tasks of approximately six hours; results were written to disk.

  The other light-weight task is the final birthday paradox search, since we store only a few keys in a hash table.
- *Medium memory (up to 4 GB).* These are the final computations modulo ℓ, for $\ell \leq 23$ (interpolation of resultants, parasite removal, finding (s_1, s_2)), and the computations modulo prime powers.
- *High memory.* These are the final computations modulo ℓ, for $\ell = 29$ and $\ell = 31$: due to the large degrees of the polynomials we handle, memory can become a bottleneck. Specific optimizations were needed to fit these computations into the RAM of the Bull machines.

We added dependencies between tasks. Some of them are due to the feasibility of the computation (for a given ℓ, the computation of all required resultants and subresultants must come before the rest of the computation); some other dependencies help us save computations: it is important not to start the computation for a prime ℓ before the computation for the previous small primes is finished. Indeed, one may discover that one of the two group orders is 0 modulo one of the previous primes, so that the rest of the computation is useless. In the same spirit, it is suboptimal to start the computations modulo powers of 2, 3, 5, 7, before having completed computations modulo all the primes. We remark however that we ran many lifting computations before being sure that they were really necessary, in order to tune our software, and make the best use of the clusters.

We wrote Python and shell scripts that handle these tasks, based on dependencies and resource availability, and ensure that on a Whale node, at most one medium memory task will run, and no high memory task. Medium tasks are given a high priority: most of the time, a 4-core node of Whale gets one of these tasks and three tiny memory tasks, so its memory is well utilized. High memory tasks are sent to Bull, and the results are centralized on Whale; the amount of communication between the clusters is very low compared to the computation time.

Statistics. We started with all possible squares of Theta constants between -40 and 40.

- Eliminating those that correspond to a degenerate Kummer surface, those that do not satisfy the rationality conditions, those for which the Jacobian is $(2,2)$-decomposable, and keeping only one choice per isomorphism class, there are 82639 remaining candidates.
- Among them, there are 35525 for which the group orders are not 0 modulo 3.
- Among them, there are 21201 for which the group orders are not 0 modulo 5.
- Among them, there are 5038 for which the group orders are 16 modulo 32.
- Among them, there are 3608 for which the group orders are not 0 modulo 7. These survivors were sent to the SHARCNET cluster.

During the modulo ℓ computations, for $\ell = 11, 13, 17, 19, 23, 29, 31$, 1214 candidates were found to have group order zero modulo ℓ, and thus aborted. In total, 586 curves were fully counted: among them, 48 gave a Jacobian or the Jacobian of the twist with a suitable group order, and only one curve was twist-secure. The remaining curves were not fully counted: we stopped our computation soon after having found the winning curve.

It takes on the order of 1000 CPU hours to complete the point-counting for a single curve. Working with the ℓ-torsion for $\ell = 11, 13, 17, 19, 23, 29, 31$ gives us (s_1, s_2) modulo $955049953 \simeq 2^{30}$; this is of course not enough to reconstruct (s_1, s_2) uniquely. Tables 7 to 10 show what further information can be deduced from torsion lifting over these curves. We represent this information in base 2, to give a uniform overview (we use degree bounds to stop the lifting; the number of curves appearing in these tables are not all the same for all ℓ's, due to early abort phenomena).

In most of the cases we finished the computation using the two-dimensional birthday paradox algorithm of Gaudry and Schost (2004a), running at about 285,000 iterations per second per core. In about 10% of the cases, we detected that the information we had modulo small prime powers was good enough to start the birthday paradox computation after $\ell = 29$. In a few cases, we missed it and we were able to obtain s_1 exactly; then s_2 was computed very quickly using a low-memory one-dimensional birthday paradox algorithm: we spent more time than necessary in the modular computations for these curves.

Finally, we mention that the various genericity assumptions we made throughout the paper (typically, that the divisors we were looking for had weight 2, etc) were satisfied in *all* the experiments we did. This is maybe not surprising, since we expect that these assumptions are satisfied in codimension 1 in the space of all curves, and since our base field has such a large cardinality.

precision on (s_1, s_2)	number of curves
$(2^{10}, 2^{12})$	1
$(2^{11}, 2^{13})$	3
$(2^{12}, 2^{14})$	3
$(2^{13}, 2^{15})$	49
$(2^{14}, 2^{16})$	445
$(2^{15}, 2^{17})$	182

Table 7. Available information from 2^k-torsion

precision on (s_1, s_2)	number of curves
$3^3 \simeq 2^{4.8}$	1
$3^4 \simeq 2^{6.3}$	3
$3^5 \simeq 2^{7.9}$	5
$3^6 \simeq 2^{9.5}$	644
$3^7 \simeq 2^{11}$	618

Table 8. Available information from 3^k-torsion

precision on (s_1, s_2)	number of curves
$5 \simeq 2^{2.3}$	346
$5^2 \simeq 2^{4.6}$	160
$5^3 \simeq 2^7$	93
$5^4 \simeq 2^{9.3}$	51
$5^5 \simeq 2^{11.6}$	8

Table 9. Available information from 5^k-torsion

precision on (s_1, s_2)	number of curves
$7 \simeq 2^{2.8}$	437
$7^2 \simeq 2^{5.6}$	174
$7^3 \simeq 2^{8.4}$	5

Table 10. Available information from 7^k-torsion

A twist-secure curve. The curve $\mathcal{C}_{11,-22,-19,-3}$ with squared Theta constants $a^2 = 11$, $b^2 = -22$, $c^2 = -19$, $d^2 = -3$ defined over $\mathbb{F}_p$, with $p = 2^{127} - 1$, is twist-secure: it has a Jacobian group order that is 16 times a prime, and the same is true for its quadratic twist. The characteristic polynomial of the Frobenius endomorphism is $T^4 - s_1T^3 + s_2T^2 - s_1pT + p^2$, with

$$s_1 = -7393453752833430168 \quad \text{and} \quad s_2 = -5869365520420357320550202376622337941 0.$$

One gets the group orders:

$$2^4 \times 180925139433306555357191732647120652144130617439968355857167262354635672633 9$$

and

$$2^4 \times 180925139433306555341467595505029059892350884363594131307776729780117962605 1.$$

A possible hyperelliptic equation is

$$\begin{aligned} y^2 = {} & x^5 + 64408548613810695909971240431892164827\,x^4 \\ & +76637216448498510246042731975843417626\,x^3 \\ & +15473509497256504102336691809959863985 1\,x^2 \\ & +98557324435909905133349189668472772 22\,x \\ & +81689052950067229064357938692912969725. \end{aligned}$$

References

Adleman, L., Huang, M.-D., 2001. Counting points on curves and abelian varieties over finite fields. Journal of Symbolic Computation. 32, 171–189.

Aho, A., Steiglitz, K., Ullman, J. D., 1975. Evaluating polynomials at fixed sets of points. SIAM J. Comput. 4 (4), 533–539.

Bernstein, D. J., 2006a. Curve25519: new Diffie-Hellman speed records. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (Eds.), Public Key Cryptography – PKC 2006. Vol. 3958 of Lecture Notes in Comput. Sci. Springer-Verlag, pp. 207–228.

Bernstein, D. J., 2006b. Elliptic vs. hyperelliptic, part 1. Talk given at ECC 2006. Slides available at `http://cr.yp.to/talks.html#2006.09.20`.

Bostan, A., Flajolet, P., Salvy, B., Schost, É., 2006. Fast computation of special resultants. Journal of Symbolic Computation. 41, 1–29.

Bostan, A., Schost, É., 2005. Polynomial evaluation and interpolation on special sets of points. Journal of Complexity 21 (4), 420–446.

Brent, R. P., Kung, H. T., 1978. Fast algorithms for manipulating formal power series. Journal of the Association for Computing Machinery 25 (4), 581–595.

Bröker, R., Lauter, K., 2009. Modular polynomials for genus 2. LMS Journal of Computation and Mathematics 12, 326–339.

C. Pascal, Schost, E., 2006. Change of order for bivariate triangular sets. In: ISSAC'06. ACM, pp. 277–284.

Cantor, D. G., 1994. On the analogue of the division polynomials for hyperelliptic curves. J. Reine Angew. Math. 447, 91–145.

Chambert-Loir, A., 2008. Compter (rapidement) le nombre de solutions d'équations dans les corps finis. Astérisque 317, 39–90, séminaire Bourbaki 968, novembre 2006.

Cohen, H., Frey, G. (Eds.), 2005. Handbook of elliptic and hyperelliptic curve cryptography. Chapman & Hall / CRC.

Couveignes, J.-M., Lercier, R., 2009. Fast construction of irreducible polynomials over finite fields. Eprint arXiv:0905.1642v2Submitted for publication.

Dahan, X., Schost, É., 2004. Sharp estimates for triangular sets. In: ISSAC'04. ACM, pp. 103–110.

Faugère, J.-C., Lubicz, D., Robert, D., 2009. Computing modular correspondences for abelian varieties, preprint.
URL `http://hal.archives-ouvertes.fr/hal-00426338/en/`

Feo, L. D., 2010. Fast algorithms for computing isogenies between ordinary elliptic curves in small characteristic. Manuscript, `http://arxiv.org/abs/1002.2597`.

Gathen, J., Gerhard, J., 1999. Modern computer algebra. Cambridge University Press.

Gathen, J., Shoup, V., 1992. Computing Frobenius maps and factoring polynomials. Computational Complexity 2 (3), 187–224.

Gaudry, P., 2006. Algorithmes de comptage de points d'une courbe définie sur un corps fini, to appear in *Méthodes explicites en théorie des nombres*, Astérisque. `http://www.loria.fr/~gaudry/publis/pano.pdf`.

Gaudry, P., 2007. Fast genus 2 arithmetic based on Theta functions. J. of Mathematical Cryptology 1, 243–265.

Gaudry, P., Harley, R., 2000. Counting points on hyperelliptic curves over finite fields. In: Bosma, W. (Ed.), ANTS-IV. Vol. 1838 of Lecture Notes in Comput. Sci. Springer–Verlag, pp. 313–332.

Gaudry, P., Schost, É., 2004a. Construction of secure random curves of genus 2 over prime fields. In: Advances in Cryptology, Eurocrypt'04. Vol. 3027 of Lecture Notes in Computer Science. Springer, pp. 239–256.

Gaudry, P., Schost, É., 2004b. A low-memory parallel version of Matsuo, Chao and Tsujii's algorithm. In: Algorithmic Number Theory, ANTS 6. No. 3076 in Lecture Notes in Computer Science. Springer, pp. 208–222.

Gaudry, P., Schost, É., 2005. Modular equations for hyperelliptic curves. Mathematics of Computation. 74, 429–454.

Gaudry, P., Thomé, E., 2007. The mpFq library and implementing curve-based key exchanges. In: SPEED: Software Performance Enhancement for Encryption and Decryption. pp. 49–64.

Harvey, D., 2007. Kedlaya's algorithm in larger characteristic. International Mathematics Research Notices 2007, Article ID rnm095, 29 pages.

Huang, M.-D., Ierardi, D., 1998. Counting points on curves over finite fields. Journal of Symbolic Computation. 25, 1–21.

Kaltofen, E., Shoup, V., 1997. Fast polynomial factorization over high algebraic extensions of finite fields. In: ISSAC'97. ACM, pp. 184–188.

Kedlaya, K. S., 2001. Counting points on hyperelliptic curves using Monsky-Washnitzer cohomology. J. Ramanujan Math. Soc. 16 (4), 323–338.

Kedlaya, K. S., Umans, C., 2008. Fast modular composition in any characteristic. In: FOCS. IEEE, pp. 146–155.

Lange, T., 2005. Formulae for arithmetic on genus 2 hyperelliptic curves. Applicable Algebra in Engineering, Communication and Computing 15 (5), 295–328.

Lauder, A., 2004. Deformation theory and the computation of zeta functions. Proc. London Math. Soc. 88, 565–602.

Lenstra, Jr., H. W., Pila, J., Pomerance, C., 2002. A hyperelliptic smoothness test, II. Proc. London Math. Soc. 84, 105–146.

Lercier, R., 1997. Algorithmique des courbes elliptiques dans les corps finis. Thèse, École polytechnique.

Lubicz, D., Robert, D., 2010. Computing isogenies between Abelian varieties, preprint, 48 pages.
URL `http://hal.archives-ouvertes.fr/hal-00446062/en/`

Matsuo, K., Chao, J., Tsujii, S., 2002. An improved baby step giant step algorithm for point counting of hyperelliptic curves over finite fields. In: Fiecker, C., Kohel, D. (Eds.), ANTS-V. Vol. 2369 of Lecture Notes in Comput. Sci. Springer-Verlag, pp. 461–474.

Pila, J., Oct. 1990. Frobenius maps of abelian varieties and finding roots of unity in finite fields. Mathematics of Computation. 55 (192), 745–763.

Pitcher, N., 2009. Efficient point-counting on genus-2 hyperelliptic curves. Ph.D. thesis, University of Illinois at Chicago.

Reischert, D., 1997. Asymptotically fast computation of subresultants. In: ISSAC'97. ACM, pp. 233–240.

Rouillier, F., 1999. Solving zero-dimensional systems through the Rational Univariate Representation. Applicable Algebra in Engineering, Communication and Computing 9 (5), 433–461.

Satoh, T., 2000. The canonical lift of an ordinary elliptic curve over a finite field and its point counting. J. Ramanujan Math. Soc. 15, 247–270.

Shoup, V., 1994. Fast construction of irreducible polynomials over finite fields. Journal of Symbolic Computation 17 (5), 371–391.

Shoup, V., 1995. A new polynomial factorization algorithm and its implementation. Journal of Symbolic Computation 20 (4), 363–397.

Shoup, V., 1996–2010. NTL: A library for doing number theory. `http://www.shoup.net`.

Streng, M., 2010. Complex multiplication of abelian surfaces. Ph.D. thesis, Universiteit Leiden.

Sutherland, A. V., 2009. A generic approach to searching for Jacobians. Mathematics of Computation. 78, 485–507.

Trager, B. M., 1976. Algebraic factoring and rational function integration. In: SYMSAC 76. ACM, pp. 219–226.