

# ON THE EVALUATION OF MODULAR POLYNOMIALS

ANDREW V. SUTHERLAND

ABSTRACT. We present two algorithms that, given a prime  $\ell$  and an elliptic curve  $E/\mathbb{F}_q$ , directly compute the polynomial  $\Phi_\ell(j(E), Y) \in \mathbb{F}_q[Y]$  whose roots are the  $j$ -invariants of the elliptic curves that are  $\ell$ -isogenous to  $E$ . We do not assume that the modular polynomial  $\Phi_\ell(X, Y)$  is given. The algorithms may be adapted to handle other types of modular polynomials, and we consider applications to point counting and the computation of endomorphism rings. We demonstrate the practical efficiency of the algorithms by setting a new point-counting record, modulo a prime  $q$  with more than 5,000 decimal digits, and by evaluating a modular polynomial of level  $\ell = 100,019$ .

## 1. INTRODUCTION

Isogenies play a crucial role in the theory and application of elliptic curves. A standard method for identifying (and computing) isogenies uses the classical modular polynomial  $\Phi_\ell \in \mathbb{Z}[X, Y]$ , which parameterizes pairs of  $\ell$ -isogenous elliptic curves in terms of their  $j$ -invariants. More precisely, over a field  $\mathbb{F}$  of characteristic not equal to  $\ell$ , the modular equation

$$\Phi_\ell(j_1, j_2) = 0$$

holds if and only if  $j_1$  and  $j_2$  are the  $j$ -invariants of elliptic curves defined over  $\mathbb{F}$  that are related by a cyclic isogeny of degree  $\ell$ . In practical applications,  $\mathbb{F}$  is typically a finite field  $\mathbb{F}_q$ , and  $\ell$  is a prime, as we shall assume throughout. For the sake of simplicity we assume that  $q$  is prime, but this is not essential.

A typical scenario is the following: we are given an elliptic curve  $E/\mathbb{F}_q$  and wish to determine whether  $E$  admits an  $\ell$ -isogeny defined over  $\mathbb{F}_q$ , and if so, to identify one or all of the elliptic curves that are  $\ell$ -isogenous to  $E$ . This can be achieved by computing the instantiated modular polynomial

$$\phi_\ell(Y) = \Phi_\ell(j(E), Y) \in \mathbb{F}_q[Y],$$

and finding its roots in  $\mathbb{F}_q$  (if any). Each root is the  $j$ -invariant of an elliptic curve that is  $\ell$ -isogenous to  $E$  over  $\mathbb{F}_q$ , and every such  $j$ -invariant is a root of  $\phi_\ell(Y)$ .

For large  $\ell$  the main obstacle to obtaining  $\phi_\ell$  is the size of  $\Phi_\ell$ , which is  $O(\ell^3 \log \ell)$  bits; several gigabytes for  $\ell \approx 10^3$ , and many terabytes for  $\ell \approx 10^4$ , see [8, Table 1]. In practice, alternative modular polynomials that are smaller than  $\Phi_\ell$  by a large constant factor are often used, but their size grows at the same rate and this quickly becomes the limiting factor, as noted in [13, §5.2] and elsewhere. The following quote is taken from the 2009 INRIA Project-Team TANC report [35, p. 9]:

*“... computing modular polynomials remains the stumbling block for new point counting records. Clearly, to circumvent the memory problems, one would need an algorithm that directly obtains the polynomial specialized in one variable.”*

---

This work was partially supported by NSF grant DMS-1115455.

Here we present just such an algorithm (two in fact), based on the isogeny volcano approach of [8]. Our basic strategy is to compute the instantiated modular polynomial  $\phi(Y) = \Phi_\ell(j(E), Y)$  modulo many “suitable” primes  $p$  and apply the explicit Chinese remainder theorem modulo  $q$  (see §2.4 and §2.5 for a discussion of the explicit CRT and suitable primes). However, two key issues arise.

First, if we simply lift the  $j$ -invariant  $j(E)$  from  $\mathbb{F}_q \simeq \mathbb{Z}/q\mathbb{Z}$  to  $\mathbb{Z}$  and reduce the result modulo  $p$ , when we instantiate  $\Phi_\ell(j(E), Y)$  the powers of  $j(E)$  we compute may correspond to integers that are much larger than the coefficients of  $\Phi_\ell$ , forcing us to use many more CRT primes than we would otherwise need. We address this issue by instead powering in  $\mathbb{F}_q$ , lifting the powers to  $\mathbb{Z}$ , and then reducing them modulo  $p$ . This yields our first algorithm, which is well-suited to situations where  $q$  is much larger than  $\ell$ , say  $\log q \approx \ell$ , as in point-counting applications.

Second, to achieve the optimal space complexity, we must avoid computing  $\Phi_\ell \bmod p$ . Indeed, if  $\log q \approx \log \ell$ , then  $\Phi_\ell \bmod p$  will not be much smaller than  $\Phi_\ell \bmod q$ . Our second algorithm uses an online approach to avoid storing all the coefficients of  $\Phi_\ell \bmod p$  simultaneously. This algorithm is well-suited to situations where  $\log q$  is not dramatically larger than  $\log \ell$ , say  $O(\log \ell)$  or  $O(\log^2 \ell)$ . This occurs, for example, in algorithms that compute the endomorphism ring of an elliptic curve [3], or algorithms to evaluate isogenies of large degree [20].

Under the generalized Riemann hypothesis (GRH), our first algorithm has an expected running time of  $O(\ell^3 \log^3 \ell \log \log \ell)$  and uses  $O(\ell^2 \log \ell + \ell \log q)$  space, assuming  $\log q = O(\ell \log \ell)$ .<sup>1</sup> This time complexity is the same as (and in practice faster than) the time to compute  $\Phi_\ell$ , and the space complexity is reduced by up to a factor of  $\ell$ . When  $\log q \approx \ell$  the space complexity is nearly optimal, quasi-linear in the size of  $\phi_\ell$ . The second algorithm uses  $O(\ell^3 (\log q + \log \ell) \log^{1+o(1)} \ell)$  time and  $O(\ell \log q + \ell \log \ell)$  space, under the GRH. Its space complexity is optimal for  $q = \Omega(\ell)$ , and when  $\log q = O(\log^{2-\epsilon} \ell)$  its time complexity is better than the time to compute  $\Phi_\ell$ . For  $\log q \gg \log^2 \ell$  its running time becomes less attractive and the first algorithm may be preferred, or see §3.4 for a hybrid approach.

In conjunction with the SEA algorithm, the first algorithm allows us to compute the cardinality of an elliptic curve modulo a prime  $q$  with a heuristic<sup>2</sup> running time of  $O(n^4 \log^2 n \log \log n)$ , using  $O(n^2 \log n)$  space, where  $n = \log q$ . To our knowledge, all alternative approaches applicable to prime fields increase at least one of these bounds by a factor of  $n$  or more. The running time is competitive with SEA implementations that rely on precomputed modular polynomials (as can be found in Magma [4] and PARI [36]), and can easily handle much larger values of  $q$ .

As an important practical optimization, we also evaluate modular polynomials  $\phi_\ell^f(Y) = \Phi_\ell^f(f(E), Y)$  defined by modular functions  $f(z)$  other than the  $j$ -function. This includes the Weber  $f$ -function, whose modular polynomials are smaller than the classical modular polynomial by a factor of 1728 and can be computed much more quickly (by roughly the same factor). This speedup also applies to  $\phi_\ell^f$ .

To demonstrate the capability of the new algorithms, we use a modified version of the SEA algorithm to count points on an elliptic curve modulo a prime of more than 5,000 decimal digits, and evaluate a modular polynomial of level  $\ell = 100,019$  modulo a prime of more than 25,000 decimal digits.

<sup>1</sup>See Theorem 4 for a more precise bound. We write  $\log \log n$  for  $\log \log n$  throughout.

<sup>2</sup>The heuristic relates to the distribution of Elkies primes and is a standard assumption made when using the SEA algorithm (without it there is no advantage over Schoof’s algorithm).

**Acknowledgements.** I am grateful to David Harvey for his assistance with the algorithms for fast polynomial arithmetic used in the computations described in §5, and to Daniel Kane for suggesting the hybrid approach outlined in §3.4. I also thank the referees for their comments and helpful suggestions.

## 2. BACKGROUND

This section contains a brief summary of background material that can be found in standard references such as [23, 29, 30], or in the papers [8, 32], both of which exploit isogeny volcanoes using a CRT-based approach, as we do here. For the sake of brevity, we recall only the results we need, and only in the generality necessary.

To simplify the presentation, we assume throughout that  $\mathbb{F}_p$  and  $\mathbb{F}_q$  denote prime fields with  $\ell \neq p, q$ , and, where relevant, that  $q$  is sufficiently large (typically  $q > 2\ell$ ). But this assumption is not needed for our main result; Algorithms 1 and 2 work correctly for any prime  $q$  (even  $q = \ell$ ), and can be extended to handle non-prime  $q$ .

**2.1. Isogenies.** Let  $E$  be an elliptic curve defined over a field  $\mathbb{F}$ . Recall that an isogeny  $\psi : E \rightarrow \tilde{E}$  is a morphism of elliptic curves that is also a group homomorphism from  $E(\overline{\mathbb{F}})$  to  $\tilde{E}(\overline{\mathbb{F}})$ . The kernel of a nonzero isogeny is a finite subgroup of  $E(\overline{\mathbb{F}})$ , and when  $\psi$  is separable, the size of its kernel is equal to its degree. Conversely, every finite subgroup  $G$  of  $E(\overline{\mathbb{F}})$  is the kernel of a separable isogeny (defined over the fixed field of the stabilizer of  $G$  in  $\text{Gal}(\overline{\mathbb{F}}/\mathbb{F})$ ). We say that  $\psi$  is *cyclic* if its kernel is cyclic, and call  $\psi$  an  *$N$ -isogeny* when it has degree  $N$ . Note that an isogeny of prime degree  $\ell \neq \text{char}(\mathbb{F})$  is necessarily cyclic and separable.

The classical modular polynomial  $\Phi_N$  is the minimal polynomial of the function  $j(Nz)$  over the field  $\mathbb{C}(j)$ , where  $j(z)$  is the modular  $j$ -function. As a polynomial in two variables,  $\Phi_N \in \mathbb{Z}[X, Y]$  is symmetric in  $X$  and  $Y$  and has the defining property that the roots of  $\Phi_\ell(j(E), Y)$  are precisely the  $j$ -invariants of the elliptic curves  $\tilde{E}$  that are related to  $E$  by a cyclic  $N$ -isogeny. In this paper  $N = \ell$  is prime, in which case  $\Phi_\ell(X, Y)$  has degree  $\ell + 1$  in each variable.

If  $E$  is given by a short Weierstrass equation  $Y^2 = X^3 + a_4X + a_6$ , then  $\psi$  can be expressed in the form  $\psi(x, y) = (\psi_1(x), cy \frac{d}{dx} \psi_1(x))$  for some  $c \in \overline{\mathbb{F}}^*$ . When  $c = 1$  we say that  $\psi$  and its image are *normalized*. Given a finite subgroup  $G$  of  $E(\overline{\mathbb{F}})$ , a normalized isogeny with  $G$  as its kernel can be constructed using Vélú's formulae [37], along with an explicit equation for its image  $\tilde{E}$ . Conversely, suppose we are given a root  $\tilde{j} = j(\tilde{E})$  of  $\phi_\ell(Y) = \Phi_\ell(j(E), Y)$ , and also the values of  $\Phi_X(j, \tilde{j})$ ,  $\Phi_Y(j, \tilde{j})$ ,  $\Phi_{XX}(j, \tilde{j})$ ,  $\Phi_{XY}(j, \tilde{j})$ , and  $\Phi_{YY}(j, \tilde{j})$ , where  $j = j(E)$  and

$$\Phi_X = \frac{\partial}{\partial X} \Phi_\ell, \quad \Phi_Y = \frac{\partial}{\partial Y} \Phi_\ell, \quad \Phi_{XX} = \frac{\partial^2}{\partial X^2} \Phi_\ell, \quad \Phi_{XY} = \frac{\partial^2}{\partial X \partial Y} \Phi_\ell, \quad \Phi_{YY} = \frac{\partial^2}{\partial Y^2} \Phi_\ell.$$

To this data we may apply an algorithm of Elkies [11] that computes an equation for  $\tilde{E}$  that is the image of a normalized  $\ell$ -isogeny  $\psi : E \rightarrow \tilde{E}$ , along with an explicit description of its kernel: the monic polynomial  $h_\ell(X)$  whose roots are the abscissae of the non-trivial points in  $\ker \psi$ ; see [16, Alg. 27]. The quantities  $\Phi_{XX}(j, \tilde{j})$ ,  $\Phi_{XY}(j, \tilde{j})$ , and  $\Phi_{YY}(j, \tilde{j})$  are not strictly necessary; the equation for  $\tilde{E}$  depends only on  $j$ ,  $\tilde{j}$ ,  $\Phi_X(j, \tilde{j})$  and  $\Phi_Y(j, \tilde{j})$ , and we may then apply algorithms of Bostan et al. [5] to compute  $h_\ell(X)$  (and an equation for  $\psi$ ) directly from  $E$  and  $\tilde{E}$ .

**2.2. Explicit CM theory.** Recall that the endomorphism ring of an ordinary elliptic curve  $E$  over a finite field  $\mathbb{F}_p$  is isomorphic to an order  $\mathcal{O}$  in an imaginary quadratic field  $K$ . In this situation  $E$  is said to have *complex multiplication* (CM)

by  $\mathcal{O}$ . The elliptic curve  $E/\mathbb{F}_p$  is the reduction of an elliptic curve  $\hat{E}/\mathbb{C}$  that also has CM by  $\mathcal{O}$ . The  $j$ -invariant of  $\hat{E}$  generates the ring class field  $K_{\mathcal{O}}$  of  $\mathcal{O}$ , and its minimal polynomial over  $K$  is the *Hilbert class polynomial*  $H_{\mathcal{O}} \in \mathbb{Z}[X]$ , whose degree is the class number  $h(\mathcal{O})$ .<sup>3</sup> The prime  $p$  splits completely in  $K_{\mathcal{O}}$ , and  $H_{\mathcal{O}}$  splits completely in  $\mathbb{F}_p[X]$ . For  $p > 3$ , the prime  $p$  splits completely in  $K_{\mathcal{O}}$  if and only if it satisfies the norm equation  $4p = t^2 - v^2D$ , where  $D = \text{disc}(\mathcal{O})$ , and for  $D < -4$  the integers  $t = t(p)$  and  $v = v(p)$  are uniquely determined up to sign.

We define the set

$$\text{Ell}_{\mathcal{O}}(\mathbb{F}_p) = \{j(E) : E/\mathbb{F}_p \text{ with } \text{End}(E) \simeq \mathcal{O}\},$$

which consists of the roots of  $H_{\mathcal{O}}$  in  $\mathbb{F}_p$ . Let  $\iota : \mathcal{O} \hookrightarrow \text{End}(E)$  denote the normalized embedding (so  $\iota(\alpha)^*\omega = \alpha\omega$  for all  $\alpha \in \mathcal{O}$  and invariant differentials  $\omega$  on  $E$ ; c.f. [30, Prop. II.1.1]). The ideals of  $\mathcal{O}$  act on  $\text{Ell}_{\mathcal{O}}(\mathbb{F}_p)$  via isogenies as follows. Let  $\mathfrak{a}$  be an  $\mathcal{O}$ -ideal of norm  $N$ , and define  $E[\mathfrak{a}] = \cap_{\alpha \in \mathfrak{a}} \ker \iota(\alpha)$ . There is a separable  $N$ -isogeny from  $E$  to  $\tilde{E} = E/E[\mathfrak{a}]$ , and the action of  $\mathfrak{a}$  sends  $j(E)$  to  $j(\tilde{E})$ . Principal ideals act trivially, and this induces a regular action of the class group  $\text{cl}(\mathcal{O})$  on  $\text{Ell}_{\mathcal{O}}(\mathbb{F}_p)$ . Thus  $\text{Ell}_{\mathcal{O}}(\mathbb{F}_p)$  is a principal homogeneous space, a *torsor*, for  $\text{cl}(\mathcal{O})$ .

Writing the  $\text{cl}(\mathcal{O})$ -action on the left, we note that if  $\mathfrak{a}$  has prime norm  $\ell$ , then  $\Phi_{\ell}(j, [\mathfrak{a}]j) = 0$  for all  $j \in \text{Ell}_{\mathcal{O}}(\mathbb{F}_p)$ . For  $\ell$  not dividing  $v(p)$ , the polynomial  $\phi_{\ell}(Y) = \Phi_{\ell}(j, Y)$  has either one or two roots in  $\mathbb{F}_p$ , depending on whether  $\ell$  ramifies or splits in  $K$ . In the latter case, the two roots  $[\mathfrak{a}]j$  and  $[\mathfrak{a}^{-1}]j$  can be distinguished using the Elkies kernel polynomial  $h_{\ell}(X)$ , as described in [6, §5] and [17, §3].

**2.3. Polycyclic presentations.** In order to efficiently realize the action of  $\text{cl}(\mathcal{O})$  on  $\text{Ell}_{\mathcal{O}}(\mathbb{F}_p)$ , it is essential to represent elements of  $\text{cl}(\mathcal{O})$  in terms of a set of generators with small norm. We will choose  $\mathcal{O}$  so that  $\text{cl}(\mathcal{O})$  is generated by ideals of norm bounded by  $O(1)$ , via [8, Thm. 3.3], but these generators will typically not be independent. Thus as explained in [32, §5.3], we use polycyclic presentations.

Any sequence of generators  $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_k)$  for a finite abelian group  $G$  defines a polycyclic series

$$1 = G_0 \triangleleft G_1 \triangleleft \dots \triangleleft G_{k-1} \triangleleft G_k = G,$$

with  $G_i = \langle \alpha_1, \dots, \alpha_i \rangle$ , in which every quotient  $G_i/G_{i-1} \simeq \langle \alpha_i \rangle$  is necessarily cyclic. We associate to  $\boldsymbol{\alpha}$  the sequence of *relative orders*  $r(\boldsymbol{\alpha}) = (r_1, \dots, r_k)$  defined by  $r_i = |G_i : G_{i-1}|$ . Every element  $\beta \in G$  has a unique  $\boldsymbol{\alpha}$ -*representation* of the form

$$\beta = \boldsymbol{\alpha}^e = \alpha_1^{e_1} \cdots \alpha_k^{e_k} \quad (0 \leq e_i < r_i).$$

We also associate to  $\boldsymbol{\alpha}$  the matrix of power relations  $s(\boldsymbol{\alpha}) = [s_{ij}]$  defined by

$$\alpha_i^{r_i} = \alpha_1^{s_{i,1}} \alpha_2^{s_{i,2}} \cdots \alpha_{i-1}^{s_{i,i-1}} \quad (0 \leq s_{ij} < r_j),$$

with  $s_{ij} = 0$  for  $i \leq j$ .

We call  $\boldsymbol{\alpha}$ , together with  $r(\boldsymbol{\alpha})$  and  $s(\boldsymbol{\alpha})$ , a (*polycyclic*) *presentation* for  $G$ , and if all the  $r_i$  are greater than 1, we say that the presentation is *minimal*. A generic algorithm to compute a minimal polycyclic presentation is given in [32, Alg. 2.2]. Having constructed such an  $\boldsymbol{\alpha}$ , we can efficiently enumerate  $G = \text{cl}(\mathcal{O})$  (or the torsor  $\text{Ell}_{\mathcal{O}}(\mathbb{F}_q)$ , given a starting point), by enumerating  $\boldsymbol{\alpha}$ -representations.

<sup>3</sup>As in [1], we call  $H_{\mathcal{O}}$  a Hilbert class polynomial even when  $\mathcal{O}$  is not the maximal order.

**2.4. Explicit CRT.** Let  $p_1, \dots, p_n$  be primes with product  $M$ , let  $M_i = M/p_i$ , and let  $a_i M_i \equiv 1 \pmod{p_i}$ . If  $c \in \mathbb{Z}$  satisfies  $c \equiv c_i \pmod{p_i}$ , then  $c \equiv \sum_i c_i a_i M_i \pmod{M}$ . If  $M > 2|c|$ , this congruence uniquely determines  $c$ . This is the usual CRT method.

Now suppose  $M > 4|c|$  and let  $q$  be a prime (or any integer). Then we may apply the *explicit CRT mod  $q$*  [2, Thm. 3.1] to compute

$$(1) \quad c \equiv \left( \sum_i c_i a_i M_i - rM \right) \pmod{q},$$

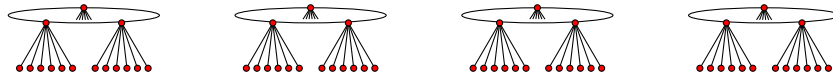
where  $r$  is the closest integer to  $\sum_i c_i a_i / p_i$ ; when computing  $r$ , it suffices to approximate each  $c_i a_i / p_i$  to within  $1/(4n)$ , by [2, Thm. 2.2].

As described in [32, §6], we may use the explicit CRT to simultaneously compute  $c \pmod{q}$  for many integers  $c$  (the coefficients of  $\phi_\ell$ , for example), using an *online algorithm*. We first precompute the  $a_i$  and  $a_i M_i \pmod{q}$ . Then, for each prime  $p_i$ , we determine the values  $c_i$  for all the coefficients  $c$  (by computing  $\phi_\ell \pmod{p_i}$ ), update two partial sums for each coefficient, one for  $\sum c_i a_i M_i \pmod{q}$  and one for  $\sum c_i a_i / p_i$ , and then discard the  $c_i$ 's. When the computations for all the  $p_i$  have been completed (these may be performed in parallel), we compute  $r$  and apply (1) for each coefficient. The space required by the partial sums is just  $O(\log q)$  bits per coefficient. See [32, §6] for further details, including algorithms for each step.

**2.5. Modular polynomials via isogeny volcanoes.** For distinct primes  $\ell$  and  $p$ , we define the *graph of  $\ell$ -isogenies*  $\Gamma_\ell(\mathbb{F}_p)$ , with vertex set  $\mathbb{F}_p$  and edges  $(j_1, j_2)$  present if and only if  $\Phi_\ell(j_1, j_2) = 0$ . Ignoring the connected components of 0 and 1728, the ordinary components of  $\Gamma_\ell(\mathbb{F}_p)$  are  *$\ell$ -volcanoes* [15, 22], a term we take to include cycles as a special case [32]. In this paper we focus on  $\ell$ -volcanoes of a particular form, for which we can compute  $\Phi_\ell \pmod{p}$  very quickly, via [8, Alg. 2.1].

Let  $\mathcal{O}$  be an order in an imaginary quadratic field  $K$  with maximal order  $\mathcal{O}_K$ , let  $\ell$  be an odd prime not dividing  $[\mathcal{O}_K : \mathcal{O}]$ , and assume  $D = \text{disc}(\mathcal{O}) < -4$ . Let  $p$  be a prime of the form  $4p = t^2 - \ell^2 v^2 D$  with  $\ell \nmid v$  and  $p \equiv 1 \pmod{\ell}$ . Then  $p$  splits completely in the ring class field of  $\mathcal{O}$ , but not in the ring class field of the order with index  $\ell^2$  in  $\mathcal{O}$ . The requirement  $p \equiv 1 \pmod{\ell}$  ensures that for  $j(E) \in \text{Ell}_{\mathcal{O}}(\mathbb{F}_p)$  we can choose  $E$  so that  $E[\ell] \subset E(\mathbb{F}_p)$ , which is critical to the efficiency of both the algorithm in [8] and our algorithms here.

The components of  $\Gamma_\ell(\mathbb{F}_p)$  that intersect  $\text{Ell}_{\mathcal{O}}(\mathbb{F}_p)$  are isomorphic  $\ell$ -volcanoes with two levels: the *surface*, whose vertices lie in  $\text{Ell}_{\mathcal{O}}(\mathbb{F}_p)$ , and the *floor*, whose vertices lie in  $\text{Ell}_{\mathcal{O}'}(\mathbb{F}_p)$ , where  $\mathcal{O}'$  is the order of index  $\ell$  in  $\mathcal{O}$ . Each vertex on the surface is connected to  $1 + \left(\frac{D}{\ell}\right) = 0, 1$  or  $2$  *siblings* on the surface, and  $\ell - \left(\frac{D}{\ell}\right)$  *children* on the floor. An example with  $\ell = 7$  and  $\left(\frac{D}{\ell}\right) = 1$  is shown below:



Provided  $h(\mathcal{O}) \geq \ell + 2$ , this set of  $\ell$ -volcanoes contains enough information to completely determine  $\Phi_\ell \pmod{p}$ . This is the basis of the algorithm in [8, Alg. 2.1], which we adapt here. Selecting a sufficiently large set of such primes  $p$  allows one to compute  $\Phi_\ell$  over  $\mathbb{Z}$  (via the CRT), or modulo an arbitrary prime  $q$  (via the explicit CRT). In order to achieve the best complexity bounds, it is important to choose both the order  $\mathcal{O}$  and the primes  $p$  carefully. We thus introduce the following definitions, in which  $c_1$  and  $c_2$  are fixed constants that do not depend on  $\ell$  or  $\mathcal{O}$  (in our implementation we used  $c_1 = 1.5$  and  $c_2 = 256$ ).

**Definition 1.** Let  $\mathcal{O}$  be a quadratic order with discriminant  $D = u^2 D_0 < 0$ , with  $D_0$  fundamental, and let  $c_1, c_2 > 1$  be constants. We say that  $\mathcal{O}$  is **suitable** for  $\ell$  if

- (i)  $\ell + 2 \leq h(\mathcal{O}) \leq c_1 \ell$ , (ii)  $4 < |D_0| \leq c_2^2$ , (iii)  $\ell^2 \leq |D| \leq c_2^2 \ell^2$ ,  
 (iv)  $\gcd(u, 2\ell D) = 1$ , (v)  $q \leq \min(c_2, \ell)$  for all primes  $q|u$ .

This definition combines the criteria in [8, Def. 4.2] and [8, Thm. 5.1]. Provided that  $c_1$  and  $c_2$  are not too small, suitable orders exist for every odd prime  $\ell$ ; with  $c_1 = 4$  and  $c_2 = 16$ , for example, we may use orders with  $D = -7 \cdot 3^{2n}$  for all  $\ell > 3$ . Ideally we want  $c_1$  to be as close to 1 as possible, but this makes it harder to find suitable orders. For the asymptotic analysis, any values of  $c_1$  and  $c_2$  will do.

**Definition 2.** A prime  $p$  is **suitable** for  $\ell$  and  $\mathcal{O}$  if  $p \equiv 1 \pmod{\ell}$  and  $p$  satisfies  $4p = t^2 - \ell^2 v^2 D$  for some  $t, v \in \mathbb{Z}$  with  $\ell \nmid v$  and  $\omega(v) \leq 2 \log(\log v + 3)$ .

The function  $\omega(v)$  counts the distinct prime divisors of  $v$ . The bound on  $\omega(v)$  ensures that if  $\mathcal{O}$  is suitable for  $\ell$  then many small primes split in  $\mathcal{O}$  and do not divide  $u$  or  $v$ . Such primes allow us to more efficiently enumerate  $\text{cl}(\mathcal{O})$  and  $\text{cl}(\mathcal{O}')$ .

**2.6. Selecting primes with the GRH.** In order to apply the isogeny volcano method to compute  $\Phi_\ell \pmod{q}$  (or  $\phi_\ell \pmod{q}$ , as we shall do), we need a sufficiently large set  $S$  of suitable primes  $p$ . We deem  $S$  to be sufficiently large whenever  $\sum_{p \in S} \log p \geq B + \log 4$ , where  $B$  is an upper bound on the logarithmic height of the integers whose reductions mod  $q$  we wish to compute with the explicit CRT. For  $\Phi_\ell(X, Y) = \sum_{i,j} a_{ij} X^i Y^j$ , we may bound  $h(\Phi_\ell) = \log \max_{i,j} |a_{ij}|$  using

$$(2) \quad h(\Phi_\ell) \leq 6\ell \log \ell + 18\ell, \quad \text{and} \quad h(\Phi_\ell) \leq 6\ell \log \ell + 16\ell + 14\sqrt{\ell} \log \ell,$$

as proved in [9] (we prefer the latter bound when  $\ell > 3187$ ).

Heuristically (and in practice), it is easy to construct the set  $S$ . Given an order  $\mathcal{O}$  of discriminant  $D$  suitable for  $\ell$ , we fix  $v = 2$  if  $D \equiv 1 \pmod{8}$  and  $v = 1$  otherwise, and for increasing  $t \equiv 2 \pmod{\ell}$  of correct parity we test whether  $p = (t^2 - v^2 \ell^2 D)/4$  is prime. We add each prime value of  $p$  to  $S$ , and stop when  $S$  is sufficiently large.

Unfortunately, we cannot prove that this method will find *any* primes, even under the GRH. Instead, we use Algorithm 6.2 in [8], which picks an upper bound  $x$  and generates random integers  $t$  and  $v$  in suitable intervals to obtain candidate primes  $p = (t^2 - v^2 \ell^2 D)/4 \leq x$  that are then tested for primality. The algorithm periodically increases  $x$ , so its expected running time is  $O(B^{1+\epsilon})$ , even without the GRH. To ensure that the bound on  $\omega(v)$  in Definition 2 is satisfied, unsuitable  $v$ 's are discarded; this occurs with negligible probability.

Under the GRH, there are effective constants  $c_3, c_4 > 0$  such that  $x \geq c_3 \ell^6 \log^4 \ell$  guarantees at least  $c_4 \ell^3 \log^3 \ell$  suitable primes less than  $x$ , by [8, Thm. 4.4]. Asymptotically, this is far more than the  $O(\ell)$  primes we need to compute  $\Phi_\ell \pmod{q}$ . Here we may consider larger values of  $B$ , and in general,  $x = O(B^2 + \ell^6 \log^4 \ell)$  suffices. We note that  $S$  contains  $O(B/\log B)$  primes (unconditionally), and under the GRH we have  $\log p = O(\log B + \log \ell)$  for all  $p \in S$ .

### 3. ALGORITHMS

Let  $q$  be a prime and let  $E$  be an elliptic curve over  $\mathbb{F}_q$ . A simple algorithm to compute  $\phi_\ell(Y) = \Phi_\ell(j(E), Y) \in \mathbb{F}_q[Y]$  with the explicit CRT works as follows. Let  $\hat{j}$  be the integer in  $[0, q-1]$  corresponding to  $j(E) \in \mathbb{F}_q \simeq \mathbb{Z}/q\mathbb{Z}$ . For a sufficiently large set  $S$  of suitable primes  $p$ , compute  $\Phi_\ell(X, Y) \pmod{p}$  using the isogeny volcano

algorithm and evaluate  $\Phi_\ell(\hat{j}, Y) \bmod p$  to obtain  $\bar{\phi}_\ell \in \mathbb{F}_p[Y]$ , and use the explicit CRT mod  $q$  to eventually obtain  $\phi_\ell \in \mathbb{F}_q[Y]$ .

This naïve algorithm suffers from two significant defects. The most serious is that we may now require a much larger set  $S$  than is needed to compute  $\Phi_\ell \bmod q$ . Compared to the coefficients of  $\Phi_\ell$ , which have height  $h(\Phi_\ell) = O(\ell \log \ell)$  bounded by (2), we now need to use the  $O(\ell \log \ell + \ell \log q)$  bound

$$(3) \quad h(\Phi_\ell(\hat{j}, Y)) \leq h(\Phi_\ell) + (\ell + 1) \log q + \log(\ell + 2),$$

since  $\Phi_\ell(\hat{j}, Y)$  involves powers of  $\hat{j}$  up to  $\hat{j}^{\ell+1}$ .

If  $\log q$  is comparable to  $\log \ell$ , then the difference between the bounds in (2) and (3) may be negligible. But when  $\log q$  is comparable to  $\ell$ , using the bound in (3) increases the running time dramatically. This issue is addressed by Algorithm 1.

The second defect of the naïve algorithm is that although its space complexity may be significantly better than the  $O(\ell^2 \log q)$  space required to compute  $\Phi_\ell \bmod q$ , it is still quasi-quadratic in  $\ell$ . But the size of  $\phi_\ell$  is linear in  $\ell$ , so we might hope to do better, and indeed we can. This is achieved by Algorithm 2.

A hybrid approach that combines aspects of both algorithms is discussed in §3.4.

**3.1. Algorithm 1.** The increase in the height bound from (2) to (3) is caused by the fact that *we are exponentiating in the wrong ring*. Rather than lifting  $j(E) \in \mathbb{F}_q$  to the integer  $\hat{j}$  and computing powers of its reduction in  $\mathbb{F}_p$  (which simulates powering in  $\mathbb{Z}$ ), we should instead compute powers  $j(E), j(E)^2, \dots, j(E)^{\ell+1}$  in  $\mathbb{F}_q$ , lift these values to integers  $\hat{x}_1, \hat{x}_2, \dots, \hat{x}_{\ell+1}$ , and work with their reductions in  $\mathbb{F}_p$ , as in [33, §4.4] (a similar strategy is used in [21]). Of course the reductions of  $\hat{x}_1, \hat{x}_2, \dots, \hat{x}_{\ell+1}$  need not correspond to powers of any particular element in  $\mathbb{F}_p$ ; nevertheless, if we simply replace each occurrence of  $X^i$  in the modular polynomial  $\Phi_\ell(X, Y) \bmod p$  with  $\hat{x}_i \bmod p$ , we achieve the same end result using a much smaller height bound.

We now present Algorithm 1 to compute  $\phi(Y) = \phi_\ell(Y) = \Phi_\ell(j(E), y)$ . If desired, the algorithm can also compute the polynomials  $\phi_X(Y) = (\partial\Phi_\ell/\partial X)(j(E), Y)$  and  $\phi_{XX}(Y) = (\partial^2\Phi_\ell/\partial X^2)(j(E), Y)$ , which may be used to compute normalized isogenies, as described in §3.8. These optional steps are shown in parentheses.

#### Algorithm 1

**Input:** An odd prime  $\ell$ , a prime  $q$ , and  $j(E) \in \mathbb{F}_q$ .

**Output:** The polynomial  $\phi(Y) = \Phi_\ell(j(E), Y) \in \mathbb{F}_q[Y]$  (and  $\phi_X(Y)$  and  $\phi_{XX}(Y)$ ).

1. Select an order  $\mathcal{O}$  suitable for  $\ell$  and a set of suitable primes  $S$  (see §2.6), using the height bound  $B = 6\ell \log \ell + 18\ell + \log q + 3 \log(\ell + 2)$ .
2. Compute the Hilbert class polynomial  $H_{\mathcal{O}}(X)$  via [32, Alg. 2].
3. Perform CRT precomputation mod  $q$  using  $S$  (see §2.4).
4. Compute integers  $\hat{x}_i \in [0, q - 1]$  such that  $\hat{x}_i \equiv j(E)^i \bmod q$ , for  $0 \leq i \leq \ell + 1$ .
5. For each prime  $p \in S$ :
  - a. Compute  $\Phi_\ell(X, Y) \bmod p$  using  $H_{\mathcal{O}}$ , via [8, Alg. 2.1].
  - b. Compute  $\bar{\phi}(Y) = \sum_{i,j} a_{ij} \hat{x}_i Y^j \bmod p$ , where  $\Phi_\ell(X, Y) = \sum_{i,j} a_{ij} X^i Y^j$ .
  - c. (Compute  $\bar{\phi}_X(Y) = \sum_{i,j} i a_{ij} \hat{x}_i Y^j \bmod p$   
and also  $\bar{\phi}_{XX}(Y) = \sum_{i,j} i(i-1) a_{ij} \hat{x}_i Y^j \bmod p$ ).
  - d. Update CRT sums for each coefficient  $c_i$  of  $\bar{\phi}$  (and of  $\bar{\phi}_X$  and  $\bar{\phi}_{XX}$ ).
6. Perform CRT postcomputation to obtain  $\phi$  (and  $\phi_X$  and  $\phi_{XX}$ ) mod  $q$ .
7. Output  $\phi$  (and  $\phi_X$  and  $\phi_{XX}$ ).

**Proposition 3.** *The output  $\phi(Y)$  of Algorithm 1 is equal to  $\Phi_\ell(j(E), Y)$ . (and  $\phi_X(Y) = (\partial\Phi_\ell/\partial X)(j(E), Y)$  and  $\phi_{XX}(Y) = (\partial^2\Phi_\ell/\partial X^2)(j(E), Y)$ ).*

*Proof.* Let  $\varphi = \Phi_\ell(\hat{j}, Y) \in \mathbb{F}_q[Y]$ . Let  $\hat{x}_i \in \mathbb{Z}$  be as in step 4. Write  $\Phi_\ell$  as  $\sum_{i,j} a_{ij} X^i Y^j$ , with  $a_{ij} \in \mathbb{Z}$  and let  $\hat{\phi} = \sum_{i,j} a_{ij} \hat{x}_i Y^j \in \mathbb{Z}[Y]$ . Then  $\varphi \equiv \hat{\phi} \pmod{q}$ , and  $\bar{\phi} \equiv \hat{\phi} \pmod{p}$ . To prove  $\phi = \varphi$ , we only need to show  $h(\hat{\phi}) \leq B$ . We have

$$|\sum_i a_{ij} \hat{x}_i| \leq (\ell + 2)q \exp h(\Phi_\ell),$$

for  $0 \leq j \leq \ell + 1$ , hence  $h(\hat{\phi}) \leq B$ . The proofs for  $\phi_X$  and  $\phi_{XX}$  are analogous. We note that the last term in  $B$  can be reduced to  $\log(\ell + 2)$  if  $\phi_X$  and  $\phi_{XX}$  are not being computed.  $\square$

**Theorem 4.** *Assume the GRH. The expected running time of Algorithm 1 is  $O(\ell^2 B \log^2 B \log B)$ , where  $B = O(\ell \log \ell + \log q)$  is as specified in step 1. It uses  $O(\ell \log q + \ell^2 \log B)$  space.*

*Proof.* We use  $M(n) = O(n \log n \log n)$  to denote the cost of multiplication [25]. For step 1, we assume the time spent selecting  $\mathcal{O}$  is negligible (as noted in §2.5, one may simply choose orders with discriminants of the form  $D = -7 \cdot 3^{2n}$ ), and under the GRH the expected time to construct  $S$  is  $O(B^{1+\epsilon})$ , using  $O(B)$  space, as explained in §2.6. Step 2 uses  $O(\ell^{2+\epsilon})$  expected time and  $O(\ell(\log \ell + \log q))$  space, by [32, Thm. 1], since  $h(D) = O(\ell)$ . An analysis as in [32, §6.3] shows that the total cost of all CRT operations is  $O(\ell M(B) \log B)$  time and  $O(\ell \log q)$  space. Step 4 uses  $O(\ell M(\log q))$  time and  $O(\ell \log q)$  space.

The set  $S$  contains  $O(B/\log B)$  primes  $p$ , and under the GRH,  $\log p = O(\log B)$ ; see §2.6. The cost per  $p$  is dominated by step 5a, which takes  $O(\ell^2 \log^3 B \log B)$  expected time and  $O(\ell^2 \log B)$  space, by [8]. This yields an  $O(\ell^2 B \log^2 B \log B)$  bound for step 5, which dominates, and the total space is  $O(\ell \log q + \ell^2 \log B)$ .  $\square$

When  $\log q = \Theta(\ell)$ , the time bound in Theorem 4 reduces to  $O(\ell^3 \log^3 \ell \log \ell)$ , the same as the time to compute  $\Phi_\ell \pmod{q}$ , and the space bound is  $O(\ell \log \ell \log q)$ , which is within an  $O(\log \ell)$  factor of the best possible.

**3.2. Algorithm 2.** We now present Algorithm 2, which for  $q > \ell$  has optimal space complexity  $O(\ell \log q)$ . When  $q$  is reasonably small, say  $\log q = o(\log^2 \ell)$ , Algorithm 2 is also faster than Algorithm 1, but when  $\log q$  is large it may be much slower, since it uses the same height bound (3) as the naïve approach (see §3.4 for a hybrid approach). The computation of  $\bar{\phi} \in \mathbb{F}_p[Y]$  is more intricate, so we present it separately as Algorithm 2.1. Unlike Algorithm 1, it is not so easy to also compute  $\phi_X$  and  $\phi_{XX}$ , but an alternative method to compute normalized isogenies using Algorithm 2 is given in §3.8.

### Algorithm 2

**Input:** An odd prime  $\ell$ , a prime  $q$ , and  $j(E) \in \mathbb{F}_q$ .

**Output:** The polynomial  $\phi(Y) = \Phi_\ell(j(E), Y) \in \mathbb{F}_q[Y]$ .

1. Select an order  $\mathcal{O}$  suitable for  $\ell$  and a suitable set of primes  $S$  (see §2.6), using the height bound  $B = 6\ell \log \ell + 18\ell + (\ell + 1) \log q + \log(\ell + 2)$ .
2. Compute the Hilbert class polynomial  $H_{\mathcal{O}}$  via [32, Alg. 2].
3. Perform precomputation for the explicit CRT mod  $q$  using  $S$ .
4. Let  $\hat{j}$  be the integer in  $[0, q - 1]$  congruent to  $j(E) \pmod{q}$ .



5. For each prime  $p \in S$ :
  - a. Compute  $\bar{\phi}(Y) = \Phi_\ell(\hat{j}, Y) \bmod p$  using  $\mathcal{O}$  and  $H_{\mathcal{O}}$  via Algorithm 2.1.
  - b. Update CRT sums for each coefficient  $c_i$  of  $\bar{\phi}$ .
6. Perform postcomputation for the explicit CRT to obtain  $\phi \in \mathbb{F}_q[X]$ .
7. Output  $\phi$ .

**Proposition 5.** *The output  $\phi(Y)$  of Algorithm 2 is equal to  $\Phi_\ell(j(E), Y)$ .*

*Proof.* This follows immediately from Proposition 8 below and the bound

$$h(\Phi_\ell(\hat{j}, Y)) = \log \max_j |\sum_i a_{ij} \hat{j}^i| \leq \log(\ell + 2) + (\ell + 1) \log q + h(\Phi_\ell) \leq B.$$

on the height of  $\Phi_\ell(\hat{j}, Y) \in \mathbb{Z}[Y]$ .  $\square$

**Theorem 6.** *Assume the GRH and that  $\log q = O(\ell^k)$  for some constant  $k$ . The expected running time of Algorithm 2 is  $O(\ell^3(\log q + \log \ell) \log \ell \log^2 \ell \log^2 \ell)$  and it uses  $O(\ell \log q + \ell \log \ell)$  space.*

*Proof.* As in the proof of Theorem 4, the expected running time is dominated by the time to compute  $\bar{\phi}(Y)$ , which by Theorem 9 is  $O(\ell^2 \log^2 p \log^2 p \log^2 p)$ . There are  $O(B/\log B)$  primes  $p \in S$ , and under the GRH we have  $\log p = O(\log B) = O(\log \ell)$ . The space complexity is dominated by the  $O(B) = O(\ell \log \ell + \ell \log q)$  size of  $S$ .  $\square$

**3.3. Algorithm 2.1.** The algorithm in [8, Alg. 2.1] computes  $\Phi_\ell \bmod p$  by enumerating the sets  $\text{Ell}_{\mathcal{O}}(\mathbb{F}_p)$  and  $\text{Ell}_{\mathcal{O}'}(\mathbb{F}_p)$ , where  $\mathcal{O}' = \mathbb{Z} + \ell\mathcal{O}$ , the latter of which contains approximately  $\ell^2$  elements. To achieve a space complexity that is quasi-linear in  $\ell$ , we cannot afford to store the entire set  $\text{Ell}_{\mathcal{O}'}(\mathbb{F}_p)$ . We must compute  $\Phi_\ell(\hat{j}, Y) \bmod p$  using an *online algorithm*, processing each  $j_k \in \text{Ell}_{\mathcal{O}'}(\mathbb{F}_p)$  as we enumerate it, and then discarding it. Let us consider how this may be done.

Let  $y_1, \dots, y_{h(\mathcal{O})}$  be the elements of  $\text{Ell}_{\mathcal{O}}(\mathbb{F}_p)$ , as enumerated using a polycyclic presentation  $\alpha$  for  $\text{cl}(\mathcal{O})$ . Each  $y_i$  is  $\ell$ -isogenous to a set  $S_i$  of *siblings* in  $\text{Ell}_{\mathcal{O}}(\mathbb{F}_p)$ , and to a set  $C_i$  of *children* in  $\text{Ell}_{\mathcal{O}'}(\mathbb{F}_p)$ ; see §2.5. Thus we have

$$\Phi_\ell(X, y_i) = \left( \prod_{\hat{j} \in S_i} (X - \hat{j}) \right) \left( \prod_{\hat{j} \in C_i} (X - \hat{j}) \right).$$

The siblings can be readily identified in our enumeration of  $\text{Ell}_{\mathcal{O}}(\mathbb{F}_p)$  using the CM action (see §2.2). To identify the children, we need to be able to determine, for any given  $j \in \mathcal{O}'$ , the set  $C_i$  in which it lies. Each  $C_i$  is a subset of the torsor  $\text{Ell}_{\mathcal{O}'}(\mathbb{F}_p)$  corresponding to a coset of the subgroup  $C \subset \text{cl}(\mathcal{O}')$  generated by the ideals of norm  $\ell^2$ ; indeed, two children have the same parent if and only if they are related by an isogeny of degree  $\ell^2$  (the composition of two  $\ell$ -isogenies).

The group  $\text{cl}(\mathcal{O}')$  acts on the cosets of  $C$ , and we need to compute this action explicitly in terms of the polycyclic presentation  $\beta$  used to enumerate  $\text{cl}(\mathcal{O}')$ . This problem is addressed by a generic group algorithm in the next section that computes a polycyclic presentation  $\gamma$  for the quotient  $\text{cl}(\mathcal{O}')/C$ , along with the  $\gamma$ -representation of the image of each generator in  $\beta$ .

As we enumerate the elements  $j_k$  of  $\text{Ell}_{\mathcal{O}'}(\mathbb{F}_p)$ , starting from a child  $j_1$  of  $y_1$  obtained via Vélú's algorithm, we keep track of the element  $\delta_k \in \text{cl}(\mathcal{O}')$  whose action sends  $j_1$  to  $j_k$ . The image of  $\delta_k$  in  $\text{cl}(\mathcal{O}')/C$  is the coset of  $C$  corresponding to the set  $C_i$  containing  $j_k$ , and we simply identify  $C_i$  with the  $i$ th element of  $\text{cl}(\mathcal{O}')/C$  as enumerated by  $\gamma$  (in the lexicographic ordering of  $\gamma$ -representations).

Thus we can compute the polynomials  $\phi_i(X) = \Phi_\ell(X, y_i)$  as we enumerate  $\text{Ell}_{\mathcal{O}'}(\mathbb{F}_p)$  by accumulating a partial product of linear factors for each  $\phi_i$ . But

since our goal is to evaluate  $z_i = \phi_i(\hat{j}) \bmod p$ , we simply substitute  $x = \hat{j} \bmod p$  into each linear factor, as we compute it, and accumulate the partial product in  $z_i$ .

Having computed the values  $z_i$  for  $1 \leq i \leq \ell + 2$ , we interpolate the unique polynomial  $\phi(Y)$  of degree at most  $\ell + 1$  for which  $\phi(y_i) = z_i$ , using Lagrange interpolation. This polynomial must be  $\Phi_\ell(\hat{j}, Y)$ . We now give the algorithm.

**Algorithm 2.1**

**Input:** An odd prime  $\ell$ , a suitable order  $\mathcal{O}$ , a suitable prime  $p$ , and  $x \in \mathbb{F}_p$ .

**Output:** The polynomial  $\phi(Y) = \Phi_\ell(x, Y) \in \mathbb{F}_p[Y]$ .

1. Compute presentations  $\alpha$  of  $\text{cl}(\mathcal{O})$  and  $\beta$  of  $\text{cl}(\mathcal{O}')$  suitable for  $p$ .
2. Represent generators of the subgroup  $C \subset \text{cl}(\mathcal{O}')$  defined above in terms of  $\beta$ .
3. Compute the presentation  $\gamma$  of  $\text{cl}(\mathcal{O}')/C$  derived from  $\beta$ , via Algorithm 3.
4. Find a root  $w_1$  of  $H_{\mathcal{O}} \bmod p$  (compute  $H_{\mathcal{O}} \bmod p$  if needed).
5. Enumerate  $\text{Ell}_{\mathcal{O}}(\mathbb{F}_p)$  as  $w_1, w_2, \dots, w_{h(\mathcal{O})}$  using  $\alpha$ .
6. Obtain  $j_1 \in \text{Ell}_{\mathcal{O}'}(\mathbb{F}_p)$  from  $w_1$  using Vélú's algorithm.
7. Set  $z_i \leftarrow 1$  and  $y_i \leftarrow \text{null}$  for  $1 \leq i \leq \ell + 2$ .
8. For each  $j_k = \delta_k j_1$  in  $\text{Ell}_{\mathcal{O}'}(\mathbb{F}_p)$  enumerated using  $\beta$ :
  - a. Compute the index  $i$  of  $\delta_k$  in the  $\gamma$ -enumeration of  $\text{cl}(\mathcal{O}')/C$ .  
If  $i > \ell + 2$  then proceed to the next  $j_k$ , skipping steps b and c below.
  - b. If  $y_i = \text{null}$  then set  $y_i$  to the  $\ell$ -parent of  $j_k$  (via Vélú's algorithm) and for each  $\ell$ -sibling  $\tilde{j}$  of  $y_i$  in  $\text{Ell}_{\mathcal{O}}(\mathbb{F}_p)$  set  $z_i \leftarrow z_i(x - \tilde{j})$ .
  - c. Set  $z_i \leftarrow z_i(x - j_k)$ .
9. Interpolate  $\phi \in \mathbb{F}_p[Y]$  such that  $\deg \phi \leq \ell + 1$  and  $\phi(y_i) = z_i$  for  $1 \leq i \leq \ell + 2$ .
10. Output  $\phi$ .

The value `null` assigned to  $y_i$  in step 7 is used to indicate that the value of  $y_i$  is not yet known. Each  $y_i$  is eventually set to a distinct  $w_j \in \text{Ell}_{\mathcal{O}}(\mathbb{F}_p)$ .

**Remark 7.** *In practical implementations, Algorithm 2 selects the primes  $p \in S$  so that the presentations  $\alpha$ ,  $\beta$ , and  $\gamma$  are the same for every  $p$  and precomputes them (the only reason they might not be the same is the presence of prime ideals whose norm divides  $v = v(p)$ , but in practice we fix  $v \leq 2$ , as discussed in §2.6).*

**Proposition 8.** *Algorithm 2.1 outputs  $\phi(Y) = \Phi_\ell(x, Y) \bmod p$ .*

*Proof.* Let  $\varphi(Y) = \Phi_\ell(x, Y)$ . It follows from the discussion above that Algorithm 2.1 computes  $z_i = \Phi_\ell(x, y_i)$  for  $1 \leq i \leq \ell + 2$ . Thus  $\phi(y_i) = z_i = \varphi(y_i)$  for  $\ell + 2$  values  $y_i \in \text{Ell}_{\mathcal{O}}(\mathbb{F}_p)$ , and these values are necessarily distinct. The polynomials  $\phi$  and  $\varphi$  both have degree at most  $\ell + 1$ , therefore  $\phi = \varphi$ .  $\square$

**Theorem 9.** *Assume the GRH. Algorithm 2.1 runs in  $O(\ell^2 n^2 \log^2 n \log^2 n)$  expected time using  $O(\ell n)$  space, where  $n = \log p$ .*

*Proof.* The time complexity is dominated by step 8, which enumerates the  $O(\ell^2)$  elements of  $\text{Ell}_{\mathcal{O}'}(\mathbb{F}_p)$  using  $\beta$ . By [8, Thm. 5.1] and the suitability of  $\mathcal{O}$  and  $p$ , we may assume each  $\beta_i = [\mathfrak{b}_i]$ , where  $\mathfrak{b}_i$  has prime norm  $b_i = O(\log n \log n)$ . Using Kronecker substitution and probabilistic root-finding [38], the expected time to find the (at most 2) roots of  $\Phi_{\mathfrak{b}_i}(j_k, Y)$  is  $O(nM(n \log n \log n))$ , which dominates the cost for each  $j_k$ . Applying  $M(n) = O(n \log n \log n)$  and multiplying by  $\ell^2$  yields the desired time bound. Taking into account  $h(\mathcal{O}) = O(\ell)$  and  $p > \ell$ , the computation of  $H_{\mathcal{O}} \bmod p$  uses  $O(\ell n)$  space, by [32, Thm. 1], and this bounds the total space.  $\square$

**3.4. A hybrid approach.** Algorithm 2 achieves an essentially optimal space complexity, but its time complexity is attractive only when  $\log q$  is not too large, say  $\log q = O(\log^2 \ell)$ . Algorithm 1 has an excellent time complexity, but achieves an optimal space complexity only when  $\log q$  is very large, say  $\log q = \Omega(\ell \log \ell)$ . To address the intermediate range, we present a hybrid approach suggested by Daniel Kane that has the same space complexity as Algorithm 2 and a time complexity that is within a polylogarithmic factor of the time complexity of Algorithm 1.

The strategy is to replace the computation of  $\bar{\phi}(Y) = \sum_{i,j} a_{ij} \hat{x}_i Y^j \pmod p$  in step 5 of Algorithm 1 with Algorithm 2.2 below. Algorithm 2.2 is similar to Algorithm 2.1, but rather than accumulating  $\ell + 2$  values  $z_i$  in parallel, we compute them individually by enumerating each of the sets  $C_i$  of children  $y_i$  in turn.

**Algorithm 2.2**

**Input:** An odd prime  $\ell$ , suitable order  $\mathcal{O}$ , suitable prime  $p$ , and  $x_1, \dots, x_{\ell+1} \in \mathbb{F}_p$ .

**Output:**  $\phi(Y) = \sum_{i,j} a_{ij} x_i Y^j \in \mathbb{F}_p[Y]$ , where  $\Phi_\ell(X, Y) = \sum_{i,j} a_{ij} X^i Y^j$ .

1. Compute presentations  $\alpha$ ,  $\beta$ , and  $\gamma$  as in Algorithm 2.1.
2. Find a root  $y_1$  of  $H_{\mathcal{O}} \pmod p$  (compute  $H_{\mathcal{O}} \pmod p$  if needed).
3. Enumerate  $\text{Ell}_{\mathcal{O}}(\mathbb{F}_q)$  as  $y_1, y_2, \dots, y_{h(\mathcal{O})}$  using  $\alpha$ .
4. Obtain  $j_1 \in \text{Ell}_{\mathcal{O}'}(\mathbb{F}_q)$  from  $y_1$  using Vélú's algorithm.
5. For  $i$  from 1 to  $\ell + 2$  do the following:
  - a. Use  $\alpha$  to compute the set  $S_i$  of siblings of  $y_i$  in  $\text{Ell}_{\mathcal{O}}(\mathbb{F}_p)$ .
  - b. Use  $\beta$  and  $\gamma$  to compute the set  $C_i$  of children of  $y_i$  in  $\text{Ell}_{\mathcal{O}'}(\mathbb{F}_p)$  (see below).
  - c. Compute  $\phi_i(X) = \prod_{\tilde{j} \in S_i} (X - \tilde{j}) \prod_{\tilde{j} \in C_i} (X - \tilde{j}) = \sum c_{ik} X^k \pmod p$ .
  - d. Compute  $z_i = \sum_k c_{ik} x_k$ .
6. Interpolate  $\phi \in \mathbb{F}_p[Y]$  such that  $\deg \phi \leq \ell + 1$  and  $\phi(y_i) = z_i$  for  $1 \leq i \leq \ell + 2$ .
7. Output  $\phi$ .

To compute the set  $C_i$  in step 5b, for each  $\tilde{j} \in C_i$  we determine the  $\delta \in \text{cl}(\mathcal{O}')$  for which  $\tilde{j} = \delta j_1$ . Under the GRH, it follows from [10, Thm. 2.1] that we can express  $\delta$  in the form  $\delta = [\mathfrak{p}_1 \cdots \mathfrak{p}_t]$ , where the ideals  $\mathfrak{p}_i$  have prime norms bounded by  $\log^c \ell$ , for any  $c > 2$ , with  $t = O(\log \ell)$ . Assuming  $\log p = O(\log \ell)$ , this implies that we can compute each  $\tilde{j}$  in  $O(\log^{6+\epsilon})$  expected time, for any  $\epsilon > 0$ .

**Proposition 10.** *Algorithm 2.2 outputs  $\phi(Y) = \sum_{i,j} a_{ij} x_i Y^j$ .*

*Proof.* Let  $\varphi(y) = \sum_{i,j} a_{ij} x_i Y^j$ . The roots of  $\phi_i(X)$  are the roots of  $\Phi_\ell(X, y_i)$ , thus  $\sum_k c_{ik} X^k = \sum_{k,j} a_{kj} X^k y_i^j$ , and we have  $c_{ik} = \sum_j a_{kj} y_i^j$ . It follows that  $\phi(y_i) = z_i = \sum_k \sum_j a_{kj} x_k y_i^j = \varphi(y_i)$ . Since  $\phi(Y)$  and  $\varphi(Y)$  both have degree at most  $\ell + 1$  and agree at  $\ell + 2$  distinct values  $y_i$ , they must be equal.  $\square$

**Theorem 11.** *Assume the GRH and that  $\log q = O(\ell \log \ell)$ . If Algorithm 1 uses Algorithm 2.2 to compute  $\bar{\phi}(Y)$  in step 5, its expected running time is  $O(\ell^3 \log^{6+\epsilon} \ell)$  using  $O(\ell \log q + \ell \log \ell)$  space.*

*Proof.* It suffices to show that if  $\log p = O(\log \ell)$ , then Algorithm 2.2 runs in  $O(\ell^2 \log^{6+\epsilon} \ell)$  expected time using  $O(\ell \log \ell)$  space. The space bound is clear. For the time bound, the cost of step 5b is  $O(\ell \log^{6+\epsilon} \ell)$  (see above), and this yields an  $O(\ell^2 \log^{6+\epsilon} \ell)$  bound on the expected time for step 5, which dominates.  $\square$

The extra logarithmic factors make the hybrid approach significantly slower than Algorithm 1 in practice, but it does allow us to achieve an essentially optimal space complexity with a quasi-cubic running time across the entire range of parameters.

**3.5. Computing a polycyclic presentation for a quotient group.** We now give a generic algorithm to derive a polycyclic presentation  $\gamma$  for a quotient of finite abelian groups  $G/H$ . This presentation can be used to efficiently compute in  $G/H$ , and to compute the image of elements of  $G$ , as required by Algorithm 2.1.

**Algorithm 3**

**Input:** A minimal polycyclic presentation  $\beta = (\beta_1, \dots, \beta_k)$  for a finite abelian group  $G$  and a subgroup  $H = \langle \alpha_1, \dots, \alpha_t \rangle$ , with each  $\alpha_i$  specified in terms of  $\beta$ .

**Output:** A polycyclic presentation  $\gamma$  for  $G/H$ , with  $\gamma_i = [\beta_i]$  for each  $\beta_i \in \beta$ .

1. Derive a polycyclic presentation  $\alpha$  for  $H$  from  $\alpha_1, \dots, \alpha_t$  via [32, Alg. 2.2].
2. Enumerate  $H$  using  $\alpha$  and create a lookup table  $T_H$  to test membership in  $H$ .
3. Derive a polycyclic presentation  $\gamma$  for  $G/H$  from  $[\beta_1], \dots, [\beta_k]$  via [32, Alg. 2.2], using  $T_H$  as described below.
4. Output  $\gamma$ , with relative orders  $r(\gamma)$  and relations  $s(\gamma)$ .

The polycyclic presentation  $\gamma$  output by Algorithm 3 is not necessarily minimal. It can be converted to a minimal presentation by simply removing those  $\gamma_i$  with  $r(\gamma_i) = 1$ , however, for the purpose of computing the image in  $G/H$  of elements of  $G$  represented in terms of  $\beta$ , it is better not to do so.

The algorithm in [32, Alg. 2.2] requires a TABLELOOKUP function that searches for a given group element in a table of distinct group elements. In Algorithm 3 above, the elements of  $G$  are uniquely represented by their  $\beta$ -representations, but elements of  $G/H$  are represented as equivalence classes  $[\delta]$ , with  $\delta \in G$ , which is not a unique representation. To implement the TABLELOOKUP function for  $G/H$ , we do the following: given  $[\delta_0] \in G/H$  and a table  $T_{G/H}$  of distinct elements  $[\delta_i] \in G/H$ , we test whether  $\delta_0 \delta_i^{-1} \in H$ , for each  $[\delta_i] \in T$ . With a suitable implementation of  $T_H$  (such as a hash table or balanced tree), membership in  $H$  can be tested in  $O(\log |G|)$  time, which is dominated by the  $O(\log^2 |G|)$  time to compute  $\delta_0 \delta_i^{-1}$ .

The problem of uniquely representing elements of  $G/H$  is solved once Algorithm 3 completes: every element of  $G/H$  has a unique  $\gamma$ -representation.

**Theorem 12.** *Algorithm 3 runs in  $O(n \log^2 n)$  time using  $O((m+n/m) \log n)$  space, where  $n = |G|$  and  $m = |H|$ .*

*Proof.* The time complexity is dominated by the  $n/m$  calls to the TABLELOOKUP function performed by [32, Alg. 2.2] in step 3, each of which performs  $m$  operations in  $G$  (using  $\beta$ -representations) and  $m$  lookups in  $T_H$ , yielding a total cost of  $O(n \log^2 n)$ . The space bound is the size of  $T_H$  plus the size of  $T_{G/H}$ .  $\square$

**3.6. Other modular functions.** For a modular function  $g$  of level  $N$  and a prime  $\ell \nmid N$ , the modular polynomial  $\Phi_\ell^g$  is the minimal polynomial of the function  $g(\ell z)$  over the field  $\mathbb{C}(g)$ . For suitable functions  $g$ , the isogeny volcano algorithm for computing  $\Phi_\ell(X, Y)$  can be adapted to compute  $\Phi_\ell^g(X, Y)$ , as described in [8, §7]. There are some restrictions:  $\Phi_\ell^g$  must have degree  $\ell + 1$  in both  $X$  and  $Y$ , and we require some additional constraints on the suitable orders  $\mathcal{O}$  that we use. Specifically, we require that there is a generator  $\tau$  of  $\mathcal{O}$  for which  $g(\tau)$  lies in the ring class field  $K_{\mathcal{O}}$ . In this case we say that  $g(\tau)$  is a *class invariant*, and let  $H_{\mathcal{O}}^g(X)$  denote its minimal polynomial over  $K$ ; see [7, 12, 14] for algorithms to compute  $H_{\mathcal{O}}^g(X)$ . We also require the polynomial  $H_{\mathcal{O}}^g$  to be defined over  $\mathbb{Z}$ .

With this setup, there is then a one-to-one correspondence between the roots  $j(\tau)$  of  $H_{\mathcal{O}}$  and the roots  $g(\tau)$  of  $H_{\mathcal{O}}^g$  in which  $\Psi^g(g(\tau), j(\tau)) = 0$ , where  $\Psi^g$  is

the minimal polynomial of  $g$  over  $\mathbb{C}(j)$ ; note that  $\Psi^g$  does not depend on  $\ell$  and is assumed to be given. The class group  $\text{cl}(\mathcal{O}) \simeq \text{Gal}(K_{\mathcal{O}}/K)$  acts compatibly on both sets of roots, and this allows us to compute  $\Phi_{\ell}^g$  modulo suitable primes  $p$  using essentially the same algorithm that is used to compute  $\Phi_{\ell} \bmod p$ . In particular, we can enumerate the set  $\text{Ell}_{\mathcal{O}}^g(\mathbb{F}_p) = \{x \in \mathbb{F}_p : H_{\mathcal{O}}^g(x) = 0\}$  using a polycyclic presentation  $\alpha$  for  $\text{cl}(\mathcal{O})$ , provided that we exclude from  $\alpha$  generators whose norm divides the level of  $g$ , and similarly for  $\text{Ell}_{\mathcal{O}'}^g(\mathbb{F}_p)$ , where  $\mathcal{O}' = \mathbb{Z} + \ell\mathcal{O}$ .

Thus Algorithms 1 and 2 can both be adapted to compute instantiated modular polynomials  $\phi^g(Y) = \Phi_{\ell}^g(x, Y) \bmod g$ . Some effort may be required to determine the correspondence between  $\text{Ell}_{\mathcal{O}}(\mathbb{F}_p)$  and  $\text{Ell}_{\mathcal{O}'}(\mathbb{F}_p)$  in cases where  $\Psi^g(X, j(E))$  or  $\Psi^g(g(E), Y)$  has multiple roots in  $\mathbb{F}_p$ ; this issue arises when we need to compute a child or parent using Vélú's algorithm. There are several techniques for resolving such ambiguities, see [8, §7.3] and especially [14], which explores this issue in detail.

We emphasize that the point  $x$  at which we are evaluating  $\Phi_{\ell}^g(x, Y)$  may be *any* element of  $\mathbb{F}_q$ , it need not correspond to the “ $g$ -invariant” of an elliptic curve.<sup>4</sup> This permits a very useful optimization that speeds up our original version of Algorithm 1 for computing  $\phi_{\ell}(Y) = \phi_{\ell}^j(Y)$  by a factor of at least 9, as we now explain.

**3.7. Accelerating the computation of  $\phi_{\ell}(Y)$  using  $\gamma_2$ .** Let  $\gamma_2(z)$  be the unique cube-root of  $j(z)$  with integral Fourier expansion, a modular function of level 3 that yields class invariants for  $\mathcal{O}$  whenever  $3 \nmid \text{disc}(\mathcal{O})$ . As noted in [8, §7.2], for  $\ell > 3$  the modular polynomial  $\Phi_{\ell}^{\gamma_2}$  can be written as

$$(4) \quad \Phi_{\ell}^{\gamma_2}(X, Y) = R(X^3, Y^3)Y^e + S(X^3, Y^3)XY + T(X^3, Y^3)X^2Y^{2-e},$$

with  $e = \ell + 1 \bmod 3$  and  $R, S, T \in \mathbb{Z}[X, Y]$ . We then have the identity

$$(5) \quad \Phi_{\ell} = R^3Y^e + (S^3 - 3RST)XY + TX^2Y^{2-e}.$$

When computing  $\Phi_{\ell}^{\gamma_2} \bmod p$  with the isogeny volcano algorithm, one can exploit (4) to speed up the computation by at least a factor of 3. In addition, the integer coefficients of  $\Phi_{\ell}^{\gamma_2}$  are also smaller than those of  $\Phi_{\ell}$  by roughly a factor of 3; we may use the height bound  $h(\Phi_{\ell}^{\gamma_2}) \leq 2\ell \log \ell + 8\ell$  from [8, Eq. 18].

Let us consider how we may modify Algorithm 1 to exploit (5), thereby accelerating the computation of  $\phi_{\ell}(Y) = \Phi_{\ell}(x, Y) \bmod q$ , where  $x = j(E) \in \mathbb{F}_q$ . Let  $r(Y) = R(x, Y) \bmod q$ , and similarly define  $s$  and  $t$  in terms of  $S$  and  $T$ . Rather than computing  $\Phi_{\ell} \bmod p$  in step 5a, we compute  $\Phi_{\ell}^{\gamma_2} \bmod p$  and derive  $R, S$ , and  $T$  from (4). We then compute polynomials  $\bar{r}, \bar{s}$ , and  $\bar{t} \bmod p$  instead of  $\bar{\phi}$  in step 5b. Finally, we recover  $r, s$ , and  $t \bmod q$  in step 6 via the explicit CRT and output

$$(6) \quad \phi = r^3Y^e + x(s^3 - 3rst)Y + x^2t^3Y^{2-e}$$

in step 7. Adjusting the height bound  $B$  appropriately, this yields a speedup of nearly a factor of 9. Note that we are not assuming  $x = j(E)$  has a cube-root in  $\mathbb{F}_q$ , or that  $\text{End}(E) \simeq \mathcal{O}$  satisfies  $3 \nmid \text{disc}(\mathcal{O})$ ; the identity (6) holds for all  $x$ .

We can similarly compute  $\phi_X$  and  $\phi_{XX}$ . To simplify the formulas, let us define  $U = (S^3 - 3RST)$  and  $u = U(x, Y) \bmod q$ . Define  $r'(Y) = (\frac{\partial}{\partial X}R)(x, Y)$  and  $r''(Y) = (\frac{\partial^2}{\partial X^2}R)(x, Y)$ , and similarly for  $s, t$ , and  $u$ . Note that  $u, u'$ , and  $u''$  can all be easily expressed in terms of  $r, r', r'', s, s', s'', t, t'$ , and  $t''$ . We replace the computation of  $\bar{\phi}_X$  and  $\bar{\phi}_{XX}$  in step 5c with analogous computations of  $\bar{r}', \bar{r}'', \bar{s}', \bar{s}'', \bar{t}'$ ,

<sup>4</sup>Every  $x \in \mathbb{F}_q$  is  $j(E)$  for some  $E/\mathbb{F}_q$ , and when  $E$  is ordinary,  $j(E)$  is the reduction of some  $j(\tau) = j(\hat{E})$  with  $\mathbb{Z}[\tau] = \mathcal{O} \simeq \text{End}(E)$ . But  $g(\tau)$  might not be a class invariant for this  $\mathcal{O}$ .

and  $\bar{t}'' \bmod p$ . We then obtain  $r, r', r'', s, s', s'', t, t',$  and  $t''$  via the explicit CRT mod  $q$  and apply

$$\begin{aligned}\phi_X &= 3r^2r'Y^e + (u + xu')Y + (2xt^3 + 3x^2t^2t')Y^{2-e}; \\ \phi_{XX} &= (6rr'r' + 3r^2r'')Y^e + (2u' + u'')Y \\ &\quad + (2t^3 + 12xt^2t' + 6x^2tt't' + 3x^2t^2t'')Y^{2-e}.\end{aligned}$$

**3.8. Normalized isogenies.** We now explain how Algorithms 1 and 2 may be used to compute normalized isogenies  $\psi$ , first using  $j$ -invariants, and then using  $g$ -invariants. Throughout this section  $j = j(E) \in \mathbb{F}_q$  denotes the  $j$ -invariant of a given elliptic curve  $E/\mathbb{F}_q$ , defined by  $y^2 = x^3 + Ax + B$ , and  $\phi(Y) = \Phi_\ell(j, Y)$ . We use  $\tilde{j} = j(\tilde{E})$  to denote a root of  $\phi(Y)$  in  $\mathbb{F}_q$ . Our goal is to compute an equation for the image of  $\psi: E \rightarrow \tilde{E}$ , and the kernel polynomial  $h_\ell(X)$  for  $\psi$ .

**3.8.1. Algorithm 1.** When computing  $\phi$ , we also compute the optional outputs  $\phi_X$  and  $\phi_{XX}$ , and then  $\phi_Y(Y) = \frac{d}{dY}\phi(Y)$ ,  $\phi_{YY}(Y) = \frac{d}{dY}\phi_Y(Y)$ , and  $\phi_{XY} = \frac{d}{dY}\phi_X(Y)$ . We then compute the quantities  $\Phi_*(j, \tilde{j}) = \phi_*(\tilde{j})$ , for  $*$  =  $X, Y, XX, XY, YY$ , as defined in §2.1, and apply Elkies' algorithm [16, Alg. 27] to compute  $\tilde{E}$  and  $h_\ell(X)$ .

**3.8.2. Algorithm 2.** Having computed  $\phi$  and obtained  $\tilde{j}$ , we run Algorithm 2 *again*, this time with the input  $\tilde{j}$ , obtaining  $\tilde{\phi}(Y) = \Phi_\ell(\tilde{j}, Y)$ , which we now regard as  $\tilde{\phi}(X) = \Phi_\ell(X, \tilde{j})$ , by the symmetry of  $\Phi_\ell$ . We then compute  $\Phi_X(j, \tilde{j}) = (\frac{d}{dX}\tilde{\phi})(\tilde{j})$  and  $\Phi_Y(j, \tilde{j}) = (\frac{d}{dY}\phi)(\tilde{j})$ , and the quantities

$$(7) \quad j' = \frac{18B}{A}j, \quad \tilde{j}' = \frac{-\Phi_X(j, \tilde{j})}{\ell\Phi_Y(j, \tilde{j})}j', \quad \tilde{m} = \frac{\tilde{j}'}{j}, \quad \tilde{k} = \frac{\tilde{j}'}{1728 - \tilde{j}'},$$

as in [16, Alg. 27]. The normalized equation for  $\tilde{E}$  is then  $y^2 = x^3 + \frac{\ell^4\tilde{m}\tilde{k}}{48}x + \frac{\ell^6\tilde{m}^2\tilde{k}}{864}$ , and the **fastElkies'** algorithm in [5] may be used to compute  $h_\ell(X)$ .

**3.8.3. Handling  $g$ -invariants.** We assume that  $g(E)$  is known to be a class invariant (see §3.9 below). Let  $g = g(E)$ ,  $\phi^g(Y) = \Phi_\ell^g(g, Y)$ , and let  $\tilde{g} = g(\tilde{E})$  denote a root of  $\phi^g(Y)$  in  $\mathbb{F}_q$ . In the case of Algorithm 1 we compute  $\Phi_X^g(g, \tilde{g}) = \phi_X^g(\tilde{g})$  and  $\Phi_Y^g(g, \tilde{g}) = (\frac{d}{dY}\phi^g)(\tilde{g})$ , and in the case of Algorithm 2 we make a second call with input  $\tilde{g}$  to obtain  $\tilde{\phi}^g(X) = \Phi_\ell^g(X, \tilde{g})$  as above. We then compute  $\Phi_X^g(g, \tilde{g}) = (\frac{d}{dX}\tilde{\phi}^g)(\tilde{g})$  and  $\Phi_Y^g(g, \tilde{g}) = (\frac{d}{dY}\phi^g)(\tilde{g})$ . We assume the modular equation  $\Psi_\ell^g(G, J) = 0$  relating  $g(z)$  to  $j(z)$  can be solved for  $j(z)$  (for the  $g(z)$  considered in [8],  $\deg_J \Psi^g(G, J) \leq 2$ ), and let  $F(G)$  satisfy  $\Psi_\ell^g(F(J), J) = 0$  and  $F' = \frac{d}{dG}F$ .

To compute the normalized equation for  $\tilde{E}$ , we proceed as in (7), except now

$$(8) \quad \tilde{j}' = -(\Phi_X^g(g, \tilde{g})F'(\tilde{g})) / (\ell\Phi_Y^g(g, \tilde{g})/F'(g))j'.$$

The **fastElkies'** algorithm in [5] may then be used to compute  $h_\ell$ , or, in the case of Algorithm 1, one may derive the trace of  $h_\ell$  using  $\Phi_{XX}^g(g, \tilde{g})$ ,  $\Phi_{XY}^g(g, \tilde{g})$ ,  $\Phi_{YY}^g(g, \tilde{g})$  as in §3.8.1, and compute  $h_\ell$  as usual. We omit the details.

**3.9. Verifying that  $g(E)$  is a class invariant.** Let  $E/\mathbb{F}_q$  be an elliptic curve that is not supersingular (see [34] for fast tests), with  $\text{End}(E) \simeq \mathcal{O}$ . As in §3.6, we call an element  $g(E)$  of  $\mathbb{F}_q$  a *class invariant* if (i)  $H_{\mathcal{O}}^g(X)$  splits into linear factors in the ring class field of  $\mathcal{O}$ , and (ii)  $g(E)$  is a common root of  $H_{\mathcal{O}}^g(X)$  and  $\Psi^g(X, j(E))$ .

For practical applications, we would like to determine whether  $g(E)$  is a class invariant without computing  $\mathcal{O}$  (indeed, the application may be to compute  $\mathcal{O}$ ).

This is often easy to do, at least as far as condition (i) is concerned. As noted in §3.6, (i) can typically be guaranteed by constraints involving  $D = \text{disc}(\mathcal{O})$  and the level  $N$  of  $g$ . Verifying condition (ii) is more difficult, in general, but it can be easily addressed in particular cases if we know that  $\Psi^g(X, j(E))$  either has a unique root in  $\mathbb{F}_q$  (which then must also be a root of  $H^g(\mathcal{O})$  once (i) is satisfied), or that all its roots in  $\mathbb{F}_q$  are roots of  $H^g(\mathcal{O})$ , or of  $H^{\bar{g}}(\mathcal{O})$  for some  $\bar{g}$  with  $\Phi_{\ell}^{\bar{g}} = \Phi_{\ell}^g$ . In the latter case we may not determine  $g(E)$  uniquely, but for the purposes of computing a normalized  $\ell$ -isogeny this does not matter, any choice will work.

Taking  $\gamma_2 = \sqrt[3]{j}$  as an example, condition (i) holds when  $(\frac{D}{3}) \neq 0$ , which means  $j(E)$  is on the surface of its 3-volcano and has either 0 or 2 siblings. This can be easily determined using [15] or [32, 4.1]. If we have  $q \equiv 2 \pmod{3}$ , the polynomial  $\Psi^g(X, j(E)) = X^3 - j(E)$  has a unique root  $g(E)$  in  $\mathbb{F}_q$  and (ii) also holds.<sup>5</sup>

As a second example, consider the Weber  $\mathfrak{f}$ -function, which is related to the  $j$ -function by  $\Psi^{\mathfrak{f}}(X, J) = (X^{24} - 16)^3 - X^{24}J$ . Now we require  $(\frac{D}{3}) \neq 0$  and  $(\frac{D}{2}) = 1$ . The latter is equivalent to  $j(E)$  being on the surface of its 2-volcano with 2 siblings. If we also have  $q \equiv 11 \pmod{12}$ , then  $\Psi^{\mathfrak{f}}(X, j(E))$  has exactly two roots  $\mathfrak{f}(E)$  and  $-\mathfrak{f}(E)$ , by [8, Lemma 7.3], and either may be used since  $\Phi_{\ell}^{\mathfrak{f}} = \Phi_{\ell}^{-\mathfrak{f}}$ .

For a more general solution, having verified condition (i), we may simply compute instantiated polynomials  $\phi(Y) = \Phi_{\ell}(x, Y)$  for *every* root  $x$  of  $\Psi^g(X, j(E))$  in  $\mathbb{F}_q$ . This can be done at essentially no additional cost, and we may then attempt to compute a normalized isogeny corresponding to each root  $x$ , which we validate by computing the dual isogeny (using the normalization factor  $c = \ell$  rather than 1) and checking whether the composition corresponds to scalar multiplication by  $\ell$  using randomly generated points in  $E(\mathbb{F}_q)$ . The cost of these validations is negligible compared to the cost of computing  $\phi(Y)$  for even one  $x$ .

As a final remark, we note that in applications such as point counting where one is only concerned with the isogeny class of  $E$ , in cases where condition (i) is not satisfied, one may be able to obtain an isogenous  $\tilde{E}$  for which (i) holds by simply climbing to the surface of the relevant  $\ell_0$ -volcanoes for the primes  $\ell_0|N$  (we regard  $N$  as fixed so  $\ell_0$  is small;  $\ell_0 = 2, 3$  in the examples above).

#### 4. APPLICATIONS

In this section we analyze the use of Algorithms 1 and 2 in two particular applications: point counting and computing endomorphism rings.

Recall that for an elliptic curve  $E/\mathbb{F}_q$ , an odd prime  $\ell$  is called an *Elkies prime* whenever  $\phi(Y) = \Phi_{\ell}(j(E), Y)$  has a root in  $\mathbb{F}_q$ . This holds if and only if  $t^2 - 4q$  is a square mod  $\ell$ , where  $t = q + 1 - \#E(\mathbb{F}_q)$ . It follows from the Chebotarev density theorem that the set of Elkies primes for  $E$  has density  $1/2$ . The complexity of the Schoof-Elkies-Atkin algorithm [26] for computing  $\#E(\mathbb{F}_q)$  depends critically on the number of *small* Elkies primes, specifically, the least  $L = L(E)$  for which

$$(9) \quad \sum_{\substack{\text{Elkies primes} \\ \ell \leq L(E)}} \log \ell > \log(4\sqrt{q}).$$

On average, one expects  $L \approx \log q$ , but even under the GRH the best proven bound is  $L = O(\log^{2+\epsilon} q)$ , see Appendix A of [24] by Satoh and Galbraith. This yields a complexity bound that is actually slightly *worse* than Schoof's original algorithm.

<sup>5</sup>There are techniques to handle  $q \equiv 1 \pmod{3}$ , see [7] for example, but they assume  $\mathcal{O}$  is known.

For practical purposes, the heuristic assumption  $L(E) = O(\log q)$  is often used when analyzing the complexity of the SEA algorithm. This assumption holds for almost all elliptic curves [28], but it is known to fail in infinitely many cases [27]. We instead adopt the following weaker heuristic.

**Heuristic 1.** *There exists a constant  $c$  such that for all sufficiently large  $q$  we have  $L(E) \leq c \log q \log \log q$  for every elliptic curve  $E/\mathbb{F}_q$ .*

**Theorem 13.** *Assume the GRH and Heuristic 1. Let  $E/\mathbb{F}_q$  be an elliptic curve over a prime field  $\mathbb{F}_q$  and let  $n = \log q$ . There is a Las Vegas algorithm to compute  $\#E(\mathbb{F}_q)$  that runs in  $O(n^4 \log^3 n \log \log n)$  expected time using  $O(n^2 \log n)$  space.*

*Proof.* Apply the SEA algorithm, using Algorithm 1 to compute  $\phi(Y) = \Phi_\ell(j(E), Y)$  (and also  $\phi_X$  and  $\phi_{XX}$ ), and ignore the Atkin primes, as in [28, Alg. 1], for example. There are  $O(n/\log n)$  primes in the sum (9), and under Heuristic 1, they are bounded by  $L = O(n \log n)$ . It follows from [28, Table 1] that the expected time to process each Elkies prime given  $\phi$  is  $O(n^3 \log^3 n \log^2 n)$ , which is dominated by the time to compute  $\phi$ , as is the space. The theorem then follows from Theorem 4.  $\square$

A common application of the SEA algorithm is to search for random curves of prime (or near prime) order, for use in cryptographic applications. As shown in [28], we no longer need Heuristic 1 to do this; we can assume  $L(E) = O(\log q)$  for a randomly chosen elliptic curve. Additionally, since we expect to count points on many curves ( $\approx \log q$ ), we can take advantage of *batching*, whereby we extend Algorithm 1 to take multiple inputs  $j(E_1) \in \mathbb{F}_{q_1}, \dots, j(E_k) \in \mathbb{F}_{q_k}$  and produce corresponding outputs for each (the  $\mathbb{F}_{q_i}$  may coincide, but they need not). Provided  $k = O(\log \ell)$ , this does not change the time complexity (relative to the largest  $\mathbb{F}_{q_i}$ ), since the most time-consuming steps depend only on  $\ell$ , not  $j(E)$ , and the space complexity is increased by at most a factor of  $k$ .<sup>6</sup>

Let  $E_{a,b}$  denote the elliptic curve defined by  $y^2 = x^3 + ax + b$ , and for any real number  $x > 3$ , let  $T(x)$  denote the set of all triples  $(q, a, b)$  with  $q \in [x, 2x]$  prime,  $a, b \in \mathbb{F}_q$ , and  $\#E_{a,b}$  prime. The following result strengthens [28, Thm. 3]

**Theorem 14.** *There is a Las Vegas algorithm that, given  $x$ , outputs a random triple  $(q, a, b) \in T(x)$  and the prime  $\#E_{a,b}(\mathbb{F}_q)$ , with  $q$  uniformly distributed over the primes in  $[x, 2x]$  and  $(a, b)$  uniformly distributed over the pairs  $(c, d) \in \mathbb{F}_q^2$  for which  $\#E_{c,d}(\mathbb{F}_q)$  is prime. Under the GRH, its expected running time is  $O(n^5 \log^2 n \log \log n)$  using  $O(n^2 \log^2 n)$  space, where  $n = \log x$ .*

*Proof.* We modify the algorithm in [28] to use Algorithm 1, operating on batches of  $O(\log n)$  inputs at a time. One then obtains an  $O(n^4 \log n \log \log n)$  bound on the average time to compute  $\#E_{a,b}(\mathbb{F}_q)$  for primes  $q \in [x, 2x]$ , and a space complexity of  $O(n^2 \log^2 n)$ . The theorem then follows from the proof of [28, Thm. 3].  $\square$

A second application of Algorithms 1 and 2 is in the computation of the endomorphism ring of an ordinary elliptic curve. The algorithm in [3] achieves a heuristically subexponential running time of  $L[1/2, \sqrt{3}/2]$  using  $L[1/2, 1/\sqrt{3}]$  space. Both Algorithms 1 and 2 improve the space complexity bound to  $L[1/2, 1/\sqrt{12}]$ , which is significant, since space is the limiting factor in these computations. Algorithm 2 also provides a slight improvement to the time complexity that is not visible in the

<sup>6</sup>These remarks also apply to Algorithm 2.



Task	CPU time (3.0 GHz AMD)
Compute $\phi_\ell^f(Y)$ with Algorithm 1	32 days
Compute $X^q \bmod \phi_\ell$ (using [19])	995 days
Compute $h_\ell$ using [16, Alg. 27]	3 days
Compute $Y^q$ and $X^q \bmod h_\ell, E$ using [18]	326 days
Compute the eigenvalue $\lambda_\ell$ using BSGS	22 days
	1378 days

TABLE 1. Breakdown of time spent computing  $\#E(\mathbb{F}_q)$ 

$L[\alpha, c]$  notation but may be practically useful. These remarks also apply to the algorithm in [20] for evaluating isogenies of large degree.

## 5. COMPUTATIONS

Using a modified version of the SEA algorithm incorporating Algorithm 1, we counted the number of points on the elliptic curve

$$y^2 = x^3 + 2718281828x + 3141592653,$$

modulo the 5011-digit prime  $q = 16219299585 \cdot 2^{16612} - 1$ . The algorithm ignored the Atkin primes and computed the trace of Frobenius  $t$  modulo 700 Elkies primes, the largest of which was  $\ell = 11681$ ; see [31] for details, including the exact value of  $t$ , which is too large to print here. The computation was distributed over 32 cores (3.0 GHz AMD Phenom II), and took about 6 weeks.

For  $\ell = 11681$ , the size of  $\phi_\ell^f(Y) = \Phi_\ell^f(\mathfrak{f}(E), Y)$  was under 20MB and took about two hours to compute on a single core. As can be seen in Table 1, the computation of  $\phi_\ell^f$  accounted for less than 3% of the total running time, despite being the asymptotically dominant step. This is primarily due to the use of the Weber  $\mathfrak{f}$ -invariant; with a less advantageous invariant (in the worst case, the  $j$ -invariant with the optimization of §3.7), the time spent computing  $\phi_\ell$  would have been comparable to or greater than the time spent on the remaining steps. But in any case the computation would still have been quite feasible.

To demonstrate the scalability of the algorithm, we computed  $\phi_\ell^f(Y)$  for an elliptic curve  $E/\mathbb{F}_q$ , with  $\ell = 100019$  and  $q = 2^{86243} - 1$ . Running on 32 cores (Algorithms 1 and 2 are both easily parallelized), this computation took less than a week. We note that the size of the instantiated modular polynomial  $\phi_\ell^f$  (and  $\phi_\ell$ ) is almost exactly one gigabyte, whereas the size of  $\Phi_\ell^f$  is many terabytes, and we estimate that the size of  $\Phi_\ell$  is around 20 or 30 petabytes.

## REFERENCES

1. Juliana Belding, Reinier Bröker, Andreas Enge, and Kristin Lauter, *Computing Hilbert class polynomials*, Algorithmic Number Theory Symposium–ANTS VIII (A. J. van der Poorten and A. Stein, eds.), Lecture Notes in Computer Science, vol. 5011, Springer, 2008, pp. 282–295.
2. Daniel J. Bernstein and Jonathan P. Sorenson, *Modular exponentiation via the explicit Chinese Remainder Theorem*, Mathematics of Computation **76** (2007), 443–454.
3. Gaetan Bisson and Andrew V. Sutherland, *Computing the endomorphism ring of an ordinary elliptic curve over a finite field*, Journal of Number Theory **113** (2011), 815–831.
4. W. Bosma, J.J. Cannon, C. Fieker, and A. Steel (eds.), *Handbook of Magma functions*, 2.17 ed., 2011, <http://magma.maths.usyd.edu.au/magma/handbook/>.

5. Alin Bostan, Bruno Salvy, François Morain, and Éric Schost, *Fast algorithms for computing isogenies between elliptic curves*, *Mathematics of Computation* **77** (2008), 1755–1778.
6. Reinier Bröker, *A  $p$ -adic algorithm to compute the Hilbert class polynomial*, *Mathematics of Computation* **77** (2008), 2417–2435.
7. ———,  *$p$ -adic class invariants*, *LMS Journal of Computation and Mathematics* **14** (2011), 108–126.
8. Reinier Bröker, Kristin Lauter, and Andrew V. Sutherland, *Modular polynomials via isogeny volcanoes*, *Mathematics of Computation* **81** (2012), 1201–1231.
9. Reinier Bröker and Andrew V. Sutherland, *An explicit height bound for the classical modular polynomial*, *Ramanujan Journal* **22** (2010), 293–313.
10. Andrew M. Childs, David Jao, and Vladimir Soukharev, *Constructing elliptic curve isogenies in quantum subexponential time*, 2011, preprint <http://arxiv.org/abs/1012.4019v2>.
11. Noam D. Elkies, *Elliptic and modular curves over finite fields and related computational issues*, *Computational Perspectives on Number Theory* (D. A. Buell and J. T. Teitelbaum, eds.), *Studies in Advanced Mathematics*, vol. 7, AMS, 1998, pp. 21–76.
12. Andreas Enge, *The complexity of class polynomial computation via floating point approximations*, *Mathematics of Computation* **78** (2009), 1089–1107.
13. ———, *Computing modular polynomials in quasi-linear time*, *Mathematics of Computation* **78** (2009), 1809–1824.
14. Andreas Enge and Andrew V. Sutherland, *Class invariants for the CRT method*, *Algorithmic Number Theory Symposium—ANTS IX* (G. Hanrot, F. Morain, and E. Thomé, eds.), *Lecture Notes in Computer Science*, vol. 6197, Springer-Verlag, 2010, pp. 142–156.
15. Mireille Fouquet and François Morain, *Isogeny volcanoes and the SEA algorithm*, *Algorithmic Number Theory Symposium—ANTS V* (C. Fieker and D. R. Kohel, eds.), *Lecture Notes in Computer Science*, vol. 2369, Springer, 2002, pp. 276–291.
16. Steven D. Galbraith, *Mathematics of public key cryptography*, Cambridge University Press, 2012.
17. Steven D. Galbraith, Florian Hess, and Nigel P. Smart, *Extending the GHS Weil descent attack*, *Advances in Cryptology—EUROCRYPT 2002*, *Lecture Notes in Computer Science*, vol. 2332, Springer, 2002, pp. 29–44.
18. Pierrick Gaudry and François Morain, *Fast algorithms for computing the eigenvalue in the Schoof-Elkies-Atkin algorithm*, *ISSAC '06: Proceedings of the 2006 international symposium on symbolic and algebraic computation*, 2006, pp. 109–115.
19. David Harvey, *A cache-friendly truncated FFT*, *Theoretical Computer Science* **410** (2009), 2649–2658.
20. David Jao and Vladimir Soukharev, *A subexponential algorithm for evaluating large degree isogenies*, *Algorithmic Number Theory Symposium—ANTS IX* (G. Hanrot, F. Morain, and E. Thomé, eds.), *Lecture Notes in Computer Science*, vol. 6197, Springer-Verlag, 2010, pp. 219–233.
21. Kiran S. Kedlaya and Christopher Umans, *Fast modular composition in any characteristic*, *49th Annual IEEE Symposium on Foundations of Computer Science (FOCS'08)*, 2008, 146–155.
22. David Kohel, *Endomorphism rings of elliptic curves over finite fields*, PhD thesis, University of California at Berkeley, 1996.
23. Serge Lang, *Elliptic functions*, second ed., Springer-Verlag, 1987.
24. Takakazu Satoh, *On  $p$ -adic point counting algorithms for elliptic curves over finite fields*, *Algorithmic Number Theory Symposium—ANTS V* (C. Fieker and D. R. Kohel, eds.), *Lecture Notes in Computer Science*, vol. 2369, Springer, 2002, pp. 43–66.
25. Arnold Schönhage and Volker Strassen, *Schnelle Multiplikation großer Zahlen*, *Computing* **7** (1971), 281–292.
26. René Schoof, *Counting points on elliptic curves over finite fields*, *Journal de Théorie des Nombres de Bordeaux* **7** (1995), 219–254.
27. Igor E. Shparlinski, *On the product of small Elkies primes*, 2013, <http://arxiv.org/abs/1112.3390>.
28. Igor E. Shparlinski and Andrew V. Sutherland, *On the distribution of Atkin and Elkies prime*, 2011, <http://arxiv.org/abs/1112.3390>.
29. Joseph H. Silverman, *The arithmetic of elliptic curves*, Springer, 1986.
30. ———, *Advanced topics in the arithmetic of elliptic curves*, Springer, 1999.

31. Andrew V. Sutherland, *Genus 1 point counting records over prime fields*, 2010, <http://math.mit.edu/~drew/SEArecords.html>.
32. ———, *Computing Hilbert class polynomials with the Chinese Remainder Theorem*, *Mathematics of Computation* **80** (2011), 501–538.
33. ———, *Accelerating the CM method*, *LMS Journal of Computation and Mathematics* **15** (2012), 172–204.
34. ———, *Identifying supersingular elliptic curves*, *LMS Journal of Computation and Mathematics* **15** (2012), 317–325.
35. INRIA Project-Team TANC, *2009 activity report*, 2009, <http://raweb.inria.fr/rapportsactivite/RA2009/tanc/tanc.pdf>.
36. The PARI Group, Bordeaux, *PARI/GP, version 2.4.3*, 2011, <http://pari.math.u-bordeaux.fr/>.
37. Jacques Vlu, *Isognies entre courbes elliptiques*, *Comptes Rendus Hebdomadaires des Sances de l'Acadmie des Sciences, Sries A et B* **273** (1971), 238–241.
38. Joachim von zur Gathen and Jrgen Gerhard, *Modern computer algebra*, second ed., Cambridge University Press, 2003.

DEPARTMENT OF MATHEMATICS, MASSACHUSETTS INSTITUTE OF TECHNOLOGY, CAMBRIDGE, MASSACHUSETTS 02139

*E-mail address:* `drew@math.mit.edu`