MATHEMATICS OF COMPUTATION
Volume 80, Number 273, January 2011, Pages 501–538
S 0025-5718(2010)02373-7
Article electronically published on May 17, 2010

COMPUTING HILBERT CLASS POLYNOMIALS WITH THE CHINESE REMAINDER THEOREM

ANDREW V. SUTHERLAND

Abstract. We present a space-efficient algorithm to compute the Hilbert class polynomial $H_D(X)$ modulo a positive integer P, based on an explicit form of the Chinese Remainder Theorem. Under the Generalized Riemann Hypothesis, the algorithm uses $O(|D|^{1/2+\epsilon}\log P)$ space and has an expected running time of $O(|D|^{1+\epsilon})$. We describe practical optimizations that allow us to handle larger discriminants than other methods, with $|D|$ as large as 10^{13} and $h(D)$ up to 10^6. We apply these results to construct pairing-friendly elliptic curves of prime order, using the CM method.

1. Introduction

Elliptic curves with a prescribed number of points have many applications, including elliptic curve primality proving [2] and pairing-based cryptography [31]. The number of points on an elliptic curve $E/\mathbb{F}_q$ is of the form $N = q + 1 - t$, where $|t| \leq 2\sqrt{q}$. For an ordinary elliptic curve, we additionally require $t \not\equiv 0 \bmod p$, where p is the characteristic of $\mathbb{F}_q$. We may construct such a curve via the *CM method*.

To illustrate, let us suppose $D < -4$ is a quadratic discriminant satisfying

$$4q = t^2 - v^2 D, \tag{1}$$

for some integer v, and let $\mathcal{O}$ denote the order of discriminant D. The j-invariant of the elliptic curve $\mathbb{C}/\mathcal{O}$ is an algebraic integer, and its minimal polynomial $H_D(X)$ is the *Hilbert class polynomial* for the discriminant D. This polynomial splits completely in $\mathbb{F}_q$, and its roots are the j-invariants of elliptic curves with an endomorphism ring isomorphic to $\mathcal{O}$. To construct such a curve, we reduce $H_D \bmod p$, compute a root in $\mathbb{F}_q$, and define an elliptic curve $E/\mathbb{F}_q$ with this j-invariant. Either E or its quadratic twist has N points, and we may easily determine which. For more details on constructing elliptic curves with the CM method, see [2, 13, 50].

The most difficult step in this process is obtaining H_D, an integer polynomial of degree $h(D)$ (the class number) and total size $O(|D|^{1+\epsilon})$ bits. There are several algorithms that, under reasonable heuristic assumptions, can compute H_D in quasi-linear time [5, 12, 22, 27], but its size severely restricts the feasible range of D. The bound $|D| < 10^{10}$ is commonly cited as a practical upper limit for the CM method [31, 43, 44, 68], and this already assumes the use of alternative class polynomials that are smaller (and less general) than H_D. As noted in [27], *space* is the limiting factor in these computations, not running time. But the CM method only uses $H_D \bmod p$, which is typically much smaller than H_D.

Received by the editor March 3, 2009 and, in revised form, September 10, 2009.
2010 *Mathematics Subject Classification.* Primary 11Y16; Secondary 11G15, 11G20, 14H52.

Here we present an algorithm to compute $H_D \bmod P$, for any positive integer P, using $O(|D|^{1/2+\epsilon}\log P)$ space. This includes the case where P is larger than the coefficients of H_D (for which we have accurate bounds); hence it may be used to determine H_D over $\mathbb{Z}$. Our algorithm is based on the CRT approach [1, 5, 17], which computes the coefficients of H_D modulo many "small" primes p and then applies the Chinese Remainder Theorem (CRT). As in [1], we use the explicit CRT [8, Thm. 3.1] to obtain $H_D \bmod P$, and we modify the algorithm in [5] to compute $H_D \bmod p$ more efficiently. Implementing the CRT computation as an online algorithm reduces the space required. We obtain a probabilistic algorithm to compute $H_D \bmod P$ whose output is always correct (a *Las Vegas* algorithm).

Under the Generalized Riemann Hypothesis (GRH), its expected running time is $O(|D|^{1+\epsilon})$. More precisely, we prove the following theorem.

Theorem 1. *Under the GRH, Algorithm* 2 *computes* $H_D \bmod P$ *in expected time* $O\big(|D|\log^5|D|(\log\log|D|)^4\big)$, *using* $O\big(|D|^{1/2}(\log|D|+\log P)\log\log|D|\big)$ *space.*

In addition to the new space bound, this improves the best rigorously proven time bound for computing H_D under the GRH [5, Thm. 1], by a factor of $\log^2|D|$. Heuristically, the time complexity is $O(|D|^{1/2}\log^{3+\epsilon}|D|)$. We also describe practical improvements that make the algorithm substantially faster than alternative methods when $|D|$ is large, and provide computational results for $|D|$ up to 10^{13} and $h(D)$ up to 10^6. In our largest examples the total size of H_D is many terabytes, but less than 200 megabytes are used to compute H_D modulo a 256-bit prime.

2. Overview

Let $\mathcal{O}$ be a quadratic order with discriminant $D < -4$. With the CRT approach, we must compute $H_D \bmod p$ for many primes p. We shall use primes in the set

$$\mathcal{P}_D = \{p > 3 \text{ prime} : 4p = t^2 - v^2D \text{ for some } t, v \in \mathbb{Z}_{>0}\}. \tag{2}$$

These primes split completely in the ring class field $K_{\mathcal{O}}$ of $\mathcal{O}$, split into principal ideals in $\mathbb{Q}[\sqrt{D}]$, and are norms of elements in $\mathcal{O}$; see [2, Prop. 2.3, Thm. 3.2]. For each $p \in \mathcal{P}_D$, the positive integers $t = t(p)$ and $v = v(p)$ are uniquely determined.

We first describe how to compute $H_D \bmod p$ for a prime $p \in \mathcal{P}_D$ and then explain how to obtain $H_D \bmod P$ for an arbitrary positive integer P. Let us begin by recalling a few pertinent facts from the theory of complex multiplication.

For any field F, we define the set

$$\mathrm{Ell}_{\mathcal{O}}(F) = \{j(E/F) : \mathrm{End}(E) \cong \mathcal{O}\}, \tag{3}$$

the j-invariants of elliptic curves defined over F whose endomorphism rings are isomorphic to $\mathcal{O}$. There are two possibilities for the isomorphism in (3), but as in [5] we make a canonical choice and henceforth identify $\mathrm{End}(E)$ with $\mathcal{O}$. For $j(E) \in \mathrm{Ell}_{\mathcal{O}}(F)$ and an invertible ideal $\mathfrak{a}$ in $\mathcal{O}$, let $E[\mathfrak{a}]$ denote the group of $\mathfrak{a}$-torsion points, those points annihilated by every $z \in \mathfrak{a} \subseteq \mathcal{O} \cong \mathrm{End}(E)$. We then define

$$j(E)^{\mathfrak{a}} = j(E/E[\mathfrak{a}]).$$

The map $j(E) \mapsto j(E)^{\mathfrak{a}}$ corresponds to an isogeny with kernel $E[\mathfrak{a}]$ and degree equal to the norm of $\mathfrak{a}$. This yields a group action of the ideal group of $\mathcal{O}$ on the set $\mathrm{Ell}_{\mathcal{O}}(K_{\mathcal{O}})$, and this action factors through the class group $\mathrm{cl}(\mathcal{O}) = \mathrm{cl}(D)$.

For a prime $p \in \mathcal{P}_D$, a bijection between $\mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$ and $\mathrm{Ell}_{\mathcal{O}}(K_{\mathcal{O}})$ arises from the Deuring lifting theorem; see [49, Thms. 13.12-14]. The following proposition then follows from the theory of complex multiplication.

Proposition 1. *For each prime $p \in \mathcal{P}_D$:*

1. *$H_D(X)$ splits completely over $\mathbb{F}_p$. It has $h(D)$ roots, which form $\mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$.*
2. *The map $j(E) \mapsto j(E)^{\mathfrak{a}}$ defines a free transitive action of $\mathrm{cl}(D)$ on $\mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$.*

For further background, we recommend the expositions in [23] and [60], and also the material in [49, Ch. 10] and [62, Ch. II].

Let p be a prime in $\mathcal{P}_D$. Our plan is to compute $H_D \bmod p$ by determining its roots and forming the product of the corresponding linear factors. By Proposition 1, we can obtain the roots by enumerating the set $\mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$ via the action of $\mathrm{cl}(D)$. All that is required is an element of $\mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$ to serve as a starting point. Thus we seek an elliptic curve $E/\mathbb{F}_p$ with $\mathrm{End}(E) \cong \mathcal{O}$. Now it may be that very few elliptic curves $E/\mathbb{F}_p$ have this endomorphism ring. Our task is made easier if we first look for an elliptic curve that at least has the desired Frobenius endomorphism, even if its endomorphism ring might not be isomorphic to $\mathcal{O}$.

For $j(E) \in \mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$, the Frobenius endomorphism $\pi_E \in \mathrm{End}(E) \cong \mathcal{O}$ corresponds to an element of $\mathcal{O}$ with norm p and trace t. Let us consider the set

$$\mathrm{Ell}_t(\mathbb{F}_p) = \{j(E/\mathbb{F}_p) : \mathrm{tr}(\pi_E) = t\}, \tag{4}$$

the j-invariants of all elliptic curves $E/\mathbb{F}_p$ with trace t. We may regard $j \in \mathrm{Ell}_t(\mathbb{F}_p)$ as identifying a particular elliptic curve $E/\mathbb{F}_p$ satisfying $j(E) = j$ and $\mathrm{tr}(\pi_E) = t$, since such an E is determined up to isomorphism [23, Prop. 14.19]. We have $\mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p) \subseteq \mathrm{Ell}_t(\mathbb{F}_p)$ and note that $\mathrm{Ell}_t(\mathbb{F}_p) = \mathrm{Ell}_{-t}(\mathbb{F}_p)$.

Recall that elliptic curves $E/\mathbb{F}_p$ and $E'/\mathbb{F}_p$ are isogenous over $\mathbb{F}_p$ if and only if $\mathrm{tr}(\pi_E) = \mathrm{tr}(\pi'_E)$; see [39, Thm. 13.8.4]. Given $j(E) \in \mathrm{Ell}_t(\mathbb{F}_p)$, we can efficiently obtain an isogenous $j(E') \in \mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$, provided $v(p)$ has no large prime factors.

This yields Algorithm 1. Its structure matches [5, Alg. 2], but we significantly modify the implementation of steps 1, 2, and 3.

Algorithm 1. *Given $p \in \mathcal{P}_D$, compute $H_D \bmod p$ as follows:*

1. *Search for a curve E with $j(E) \in \mathrm{Ell}_t(\mathbb{F}_p)$ (Algorithm 1.1).*
2. *Find an isogenous E' with $j(E') \in \mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$ (Algorithm 1.2).*
3. *Enumerate $\mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$ from $j(E')$ via the action of $\mathrm{cl}(D)$ (Algorithm 1.3).*
4. *Compute $H_D \bmod p$ as $H_D(X) = \prod_{j \in \mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)} (X - j)$.*

Algorithm 1.1 searches for $j(E) \in \mathrm{Ell}_t(\mathbb{F}_p)$ by sampling random curves and testing whether they have trace t (or $-t$). To accelerate this process, we sample a family of curves whose orders are divisible by m, for some suitable $m|(p + 1 \pm t)$. We select $p \in \mathcal{P}_D$ to ensure that such an m exists, and also to maximize the size of $\mathrm{Ell}_t(\mathbb{F}_p)$ relative to $\mathbb{F}_p$ (with substantial benefit).

To compute the isogenies required by Algorithms 1.2 and 1.3 we use the classical modular polynomial $\Phi_N \in \mathbb{Z}[X, Y]$, which parametrizes elliptic curves connected by a cyclic isogeny of degree N. For a prime $\ell \neq p$ and an elliptic curve $E/\mathbb{F}_p$, the roots of $\Phi_\ell(X, j(E))$ over $\mathbb{F}_p$ are the j-invariants of all curves $E'/\mathbb{F}_p$ connected to E via an isogeny of degree ℓ (an ℓ-isogeny) [71, Thm. 12.19]. This gives us a computationally explicit way to define the graph of ℓ-isogenies on the set $\mathrm{Ell}_t(\mathbb{F}_p)$.

As shown by Kohel [46], the connected components of this graph all have a particular shape, aptly described in [29] as a *volcano* (see Figure 1 in Section 4).

The curves in an isogeny volcano are naturally partitioned into one or more levels, according to their endomorphism rings, with the curves at the top level forming a cycle. Given an element of $\mathrm{Ell}_t(\mathbb{F}_p)$, Algorithm 1.2 finds an element of $\mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$ by climbing a series of isogeny volcanoes. Given an element of $\mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$, Algorithm 1.3 enumerates the entire set by walking along ℓ-isogeny cycles for various values of ℓ.

We now suppose that we have computed H_D modulo primes $p_1, \ldots, p_n$ and consider how to compute $H_D \bmod P$ for an arbitrary positive integer P, using the Chinese Remainder Theorem. In order to do so, we need an explicit bound B on the largest coefficient of H_D (in absolute value). Lemma 8 of Appendix 1 provides such a B, and it satisfies $\log B = O(|D|^{1/2+\epsilon})$.

Let $M = \prod p_i$, $M_i = M/p_i$ and $a_i \equiv M_i^{-1} \bmod p_i$. Suppose $c \in \mathbb{Z}$ is a coefficient of H_D. We know the values $c_i \equiv c \bmod p_i$ and wish to compute $c \bmod P$ for some positive integer P. We have

$$c \equiv \sum c_i a_i M_i \bmod M, \tag{5}$$

and if $M > 2B$ we can uniquely determine c. This is the usual CRT approach.

Alternatively, if M is slightly larger, say $M > 4B$, we may apply the explicit CRT (mod P) [8, Thm. 3.1] and compute $c \bmod P$ directly via

$$c \equiv \sum c_i a_i M_i - rM \bmod P. \tag{6}$$

Here r is the nearest integer to $\sum c_i a_i / p_i$. When computing r it suffices to approximate each rational number $c_i a_i / p_i$ to within $1/(4n)$.

As noted in [27], even when P is small one still has to compute $H_D \bmod p_i$ for enough primes to determine H_D over $\mathbb{Z}$, so the work required is essentially the same. The total size of the c_i over all the coefficients is necessarily as large as H_D.

However, instead of applying the explicit CRT at the end of the computation, we update the sums $\sum c_i a_i M_i \bmod P$ and $\sum c_i a_i / p_i$ as each c_i is computed and immediately discard c_i. This *online* approach reduces the space required.

We now give the complete algorithm to compute $H_D \bmod P$. When P is large we alter the CRT approach slightly as described in Section 7. This allows us to efficiently treat all P, including $P = M$, which is used to compute H_D over $\mathbb{Z}$.

Algorithm 2. *Compute $H_D \bmod P$ as follows:*

```
1. Select primes $p_1, \ldots, p_n \in \mathcal{P}_D$ with $\prod p_i > 4B$ (Algorithm 2.1).
2. Compute suitable presentations of $\mathrm{cl}(D)$ (Algorithm 2.2).
3. Perform CRT precomputation (Algorithm 2.3).
4. For each $p_i$:
   a. Compute the coefficients of $H_D \bmod p_i$ (Algorithm 1).
   b. Update CRT sums for each coefficient of $H_D$ (Algorithm 2.4).
5. Recover the coefficients of $H_D \bmod P$ (Algorithm 2.5).
```

The presentations computed by Algorithm 2.2 are used by Algorithm 1.3 to realize the action of the class group. The optimal presentation may vary with p_i (more precisely, $v(p_i)$), but often the same presentation is used for every p_i. Each presentation specifies a sequence of primes $\ell_1, \ldots, \ell_k$ corresponding to a sequence $\alpha_1, \ldots, \alpha_k$ of generators for $\mathrm{cl}(D)$ in which each α_i contains an ideal of norm ℓ_i. There is an associated sequence of integers $r_1, \ldots, r_k$ with the property that every $\beta \in \mathrm{cl}(D)$ can be expressed uniquely in the form

$$\beta = \alpha_1^{x_1} \cdots \alpha_k^{x_k},$$

with $0 \le x_i < r_i$. Algorithm 1.3 uses isogenies of degrees $\ell_1, \dots, \ell_k$ to enumerate $\mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$. Given the large size of $\Phi_\ell(X,Y)$, roughly $O(\ell^3 \log \ell)$ bits [21], it is critical that the ℓ_i are as small as possible. We achieve this by computing an optimal *polycyclic presentation* for $\mathrm{cl}(D)$, derived from a sequence of generators for $\mathrm{cl}(D)$. Under the Extended Reimann Hypothesis (ERH) we have $\ell_i \le 6\log^2|D|$, by [4]. This approach corrects an error in [5] which relies on a *basis* for $\mathrm{cl}(D)$ and fails to achieve such a bound (see Section 5.3 for a counterexample).

The rest of this paper is organized as follows:

- Section 3 describes how we find a curve with trace $\pm t$ (Algorithm 1.1) and how the primes $p_1, \dots, p_n$ are selected (Algorithm 2.1).
- Section 4 discusses isogeny volcanoes (Algorithms 1.2 and 1.3).
- Section 5 defines an optimal polycyclic presentation of $\mathrm{cl}(D)$ and gives an algorithm to compute one (Algorithm 2.2).
- Section 6 addresses the CRT computations (Algorithms 2.3, 2.4, and 2.5).
- Section 7 contains a complexity analysis and proves Theorem 1.
- Section 8 provides computational results.

Included in Section 8 are timings obtained while constructing pairing-friendly curves of prime order over finite fields of cryptographic size.

3. Finding an elliptic curve with a given number of points

Given a prime p and a positive integer $t < 2\sqrt{p}$, we seek an element of $\mathrm{Ell}_t(\mathbb{F}_p)$, equivalently, an elliptic curve $E/\mathbb{F}_p$ with either $N_0 = p+1-t$ or $N_1 = p+1+t$ points. This is essentially the problem considered in the Introduction, but since we do not yet know H_D, we cannot apply the CM method.

Instead, we generate curves at random and test whether $\#E \in \{N_0, N_1\}$, where $\#E$ is the cardinality of the group $E(\mathbb{F}_p)$. This test takes very little time, given the prime factorizations of N_0 and N_1, and does not require computing $\#E$. However, in the absence of any optimizations we expect to test many curves: $2\sqrt{p} + O(1)$, on average, for fixed p and varying t. Factoring N_0 and N_1 is easy by comparison.

For the CRT-based algorithm in [5], searching for elements of $\mathrm{Ell}_t(\mathbb{F}_p)$ dominates the computation. In the example given there, this single step takes more than 50 times as long as the entire computation of H_D using the floating-point method of [27]. Here we address this problem in detail, giving both asymptotic and constant factor improvements. In the aggregate, the improvements we suggest can reduce the time to find an element of $\mathrm{Ell}_t(\mathbb{F}_p)$ by a factor of over 100; under the heuristic analysis of Section 7.1 this is no longer the asymptotically dominant step.

These improvements are enabled by a careful selection of primes $p \in \mathcal{P}_D$, which is described in Section 3.3. Contrary to what one might assume, the smallest primes in $\mathcal{P}_D$ are not necessarily the best choices. The expected time to find an element of $\mathrm{Ell}_t(\mathbb{F}_p)$ can vary dramatically from one prime to the next, especially when one considers optimizations whose applicability may depend on N_0 and N_1. In order to motivate our selection criteria, we first consider how we may narrow the search by our choice of p, which determines $t = t(p)$ and therefore N_0 and N_1.

3.1. **The density of curves with trace $\pm t$.**

We may compute the density of $\mathrm{Ell}_t(\mathbb{F}_p)$ as a subset of $\mathbb{F}_p$ via a formula of Deuring [26]. For convenience we define

$$\rho(p,t) = \frac{H(4p - t^2)}{p} \approx \frac{\#\,\mathrm{Ell}_t(\mathbb{F}_p)}{\#\mathbb{F}_p}, \tag{7}$$

where $H(4p-t^2)$ is the Hurwitz class number (as in [18, Def. 5.3.6] or [23, p. 319]). A more precise formula uses weighted cardinalities, but the difference is negligible; see [23, Thm. 14.18] or [51] for further details.

We expect to sample approximately $1/\rho(p,t)$ random curves over $\mathbb{F}_p$ in order to find one with trace $\pm t$. When selecting primes $p \in \mathcal{P}_D$, we may give preference to primes with larger ρ-values. Doing so typically increases the average density by a factor of 3 or 4, compared to simply using the smallest primes in $\mathcal{P}_D$. It also makes N_0 and N_1 more likely to be divisible by small primes, which interacts favorably with the optimizations of the next section.

Using primes with large ρ-values improves the asymptotic results of Section 7 by an $O(\log |D|)$ factor. Effectively, we force the size of $\mathrm{Ell}_t(\mathbb{F}_p)$ to increase with p, even though the size of $\mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$ is fixed at $h(D)$. This process tends to favor primes in $\mathcal{P}_D$ for which $v(p)$ has many small factors, something we must consider when enumerating $\mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$ in Algorithm 1.3.

3.2. **Families with prescribed torsion.** In addition to increasing the density of $\mathrm{Ell}_t(\mathbb{F}_p)$ relative to $\mathbb{F}_p$, we can further accelerate our random search by sampling a subset of $\mathbb{F}_p$ in which $\mathrm{Ell}_t(\mathbb{F}_p)$ has even greater density. Specifically, we may restrict our search to a family of curves whose order is divisible by m, for some small m dividing N_0 or N_1 (ideally both). We have some control over N_0 and N_1 via our choice of $p \in \mathcal{P}_D$, and in practice we find we can easily arrange for N_0 or N_1 to be divisible by a suitable m, discarding only a constant fraction of the primes in $\mathcal{P}_D$ we might otherwise consider (making the primes that we do use slightly larger).

To generate a curve whose order is divisible by m, we select a random point on $Y_1(m)/\mathbb{F}_p$ and construct the corresponding elliptic curve. Here $Y_1(m)$ is the affine subcurve of the modular curve $X_1(m)$, which parametrizes elliptic curves with a point of order m. We do this using plane models $F_m(r,s) = 0$ that have been optimized for this purpose; see [65]. For m in the set $\{2,3,4,5,6,7,8,9,10,12\}$, the curve $X_1(m)$ has genus 0, and we obtain Kubert's parametrizations [47] of elliptic curves with a prescribed (cyclic) torsion subgroup over $\mathbb{Q}$. Working in $\mathbb{F}_p$, we may use any m not divisible p, although we typically use $m \le 40$, due to the cost of finding points on $F_m(r,s) = 0$.

We augment this approach with additional torsion constraints that can be quickly computed. For example, to generate a curve containing a point of order 132, it is much faster to generate several curves using $X_1(11)$ and apply tests for 3 and 4 torsion to each than it is to use $X_1(132)$. A table of particularly effective combinations of torsion constraints, ranked by cost/benefit ratio, appears in Appendix 2.

The cost of finding points on $F_m(r,s) = 0$ is negligible when m is small, but grows with the genus (more precisely, the gonality) of $X_1(m)$, which is $O(m^2)$, by [42, Thm. 1.1]. For $m < 23$ the gonality is at most 4 (see Table 5 in [65]), and points on $F_m(r,s)$ can be found quite quickly (especially when the genus is 0 or 1).

Provided that we select suitable primes from $\mathcal{P}_D$, generating curves with prescribed torsion typically improves performance by a factor of 10 to 20.

3.3. **Selecting suitable primes.** We wish to select primes in $\mathcal{P}_D$ that maximize the benefit of the optimizations considered in Sections 3.1 and 3.2. Our strategy is to enumerate a set of primes

$$S_z = \{p \in \mathcal{P}_D : 1/\rho(p,t(p)) \le z\} \tag{8}$$

that is larger than we need, and to then select a subset $S \subset S_z$ of the "best" primes. We require that S be large enough to satisfy

$$\sum_{p\in S} \lg p > b = \lg B + 2,$$

where B is a bound on the coefficients of $H_D(X)$, obtained via Lemma 8, and "lg" denotes the binary logarithm. We typically seek to make S_z roughly 2 to 4 times the size of S, starting with a nominal value for z and increasing it as required.

To enumerate S_z we first note that if $4p = t^2 - v^2D$ for some $p \in S_z$, then

$$\frac{1}{\rho(p,t)} = \frac{p}{H(4p-t^2)} = \frac{p}{H(-v^2D)} \leq z.$$

Hence for a given v, we may bound the $p \in S_z$ with $v(p) = v$ by

$$p \leq zH(-v^2D). \tag{9}$$

To find such primes, we seek t for which $p = (t^2 - v^2D)/4$ is a prime satisfying (9). This is efficiently accomplished by *sieving* the polynomial $t^2 - v^2D$; see [24, §3.2.6]. To bound $v = v(p)$ for $p \in S_z$, we note that $p > -v^2D/4$, hence

$$-v^2D < 4zH(-v^2D). \tag{10}$$

For fixed z, this inequality will fail once v becomes too large. If we have

$$\frac{v}{(\log\log(v+4))^2} \geq \frac{44zH(-D)}{-D}, \tag{11}$$

then (10) cannot hold, by Lemma 9 of Appendix 1.

Example. Consider the construction of S_z for $D = -108708$, for which we have $H(-D) = h(D) = 100$. We initially set z to $-D/(2H(-D)) \approx 543$. For $v = 1$ this yields the interval $[-v^2D/4, zH(-v^2D)] = [-D/4, -D/2] = [27177, 54354]$, which we search for primes of the form $(t^2 - D)/4$ by sieving $t^2 - D$ with $t \leq \sqrt{-2D}$, finding 17 such primes. For $v = 2$ we have $H(-v^2D) = 300$ and search the interval $[-D, -3D/2] = [108708, 163062]$ for primes of the form $(t^2 - 4D)/4$, finding 24 of them. For $v = 3$ we have $H(-v^2D) = 400$, and the interval $[-9D/4, -2D]$ is empty. The interval is also empty for $3 < v < 39$, and (11) applies to all $v \geq 39$.

At this point S_z is not sufficiently large, so we increase z, say by 50%, obtaining $z \approx 814$. This expands the intervals for $v = 1, 2$ and gives nonempty intervals for $v = 3, 4$, and we find an additional 74 primes. Increasing z twice more, we eventually reach $z \approx 1831$, at which point S_z contains 598 primes with total size around 11911 bits. This is more than twice $b = \lg B + 2 \approx 5943$, so we stop. The largest prime in S_z is $p = 5121289$, with $v(p) = 12$.

Once S_z has been computed, we select $S \subset S_z$ by ranking the primes $p \in S_z$ according to their cost/benefit ratio. The cost is the expected time to find a curve in $\mathrm{Ell}_t(\mathbb{F}_p)$, taking into account the density $\rho(p,t)$ and the m-torsion constraints applicable to N_0 and N_1, and the benefit is $\lg p$, the number of bits in p. Only a small set of torsion constraints are worth considering, and a table of these may be precomputed. See Appendix 2 for further details.

The procedure for selecting primes is summarized below. We assume that $h(D)$ has been obtained in the process of determining B and $b = \lg B + 2$, which allows $H(-D)$ and $\rho(p,t)$ to be easily computed (see (26) and (27) in Appendix 1).

Algorithm 2.1. *Given D, b, and parameters $k > 1$, $\delta > 0$, select $S \subset \mathcal{P}_D$:*

```
1. Let $z = -D/(2H(-D))$.
2. Compute $S_z = \{p \in \mathcal{P}_D : 1/\rho(p, t(p)) \le z\}$.
3. If $\sum_{p\in S_z} \lg p \le kb$, then set $z \leftarrow (1+\delta)z$ and go to step 2.
4. Rank the primes in $S_z$ by increasing the cost/benefit ratio as $p_1, \ldots, p_{n_z}$.
5. Let $S = \{p_1, \ldots, p_n\}$, with $n \le n_z$ minimal subject to $\sum_{p\in S} \lg p > b$.
```

In step 3 we typically use $k = 2$ or $k = 4$ (a larger k may find better primes), and $\delta = 1/2$. The complexity of Algorithm 2.1 is analyzed in Section 7, where it is shown to run in expected time $O(|D|^{1/2+\epsilon})$, under the GRH (Lemma 4). This is negligible compared to the total complexity of $O(|D|^{1+\epsilon})$ and very fast in practice.

In the $D = -108708$ example above, Algorithm 2.1 selects 313 primes in S_z, the largest of which is $p = 4382713$, with $v = 12$ and $t = 1370$. This largest prime is actually a rather good choice, due to the torsion constraints that may be applied to $N_0 = p + 1 - t$, which is divisible by 3, 4, and 11. We expect to test the orders of fewer than 40 curves for this prime, and on average we need to test about 60 curves for each prime in S, fewer than 20,000 in all.

For comparison, the example in [5, p. 294] uses the least 324 primes in $\mathcal{P}_D$, the largest of which is only 956929; but nearly 500,000 curves are tested, over 1500 per prime. The difference in running times is even greater, 0.2 seconds versus 18.5 seconds, due to optimizations in the testing algorithm of the next section.

3.4. **Testing curves.** When p is large, the vast majority of the random curves we generate will not have trace $\pm t$, even after applying the optimizations above. To quickly filter a batch of, say, 50 or 100 curves, we pick a random point P on each curve and simultaneously compute $(p+1)P$ and tP. Here we apply standard multi-exponentiation techniques to scalar multiplication in $E(\mathbb{F}_p)$, using a precomputed NAF representation; see [20, Ch. 9]. We perform the group operations in parallel to minimize the cost of field inversions, using affine coordinates as in [45, §4.1]. We then test whether $(p+1)P = \pm tP$, as suggested in [5], and if this fails to hold we reject the curve, since its order cannot be $p + 1 \pm t$.

To each curve that passes this test, we apply Algorithm TestCurveOrder. In the description below, $\mathcal{H}_p = [p+1-2\sqrt{p}, p+1+2\sqrt{p}]$ denotes the Hasse interval, and the index $s \in \{0, 1\}$ is used to alternate between E and its quadratic twist $\tilde{E}$.

Algorithm TestCurveOrder. *Given an elliptic curve $E/\mathbb{F}_p$ and factored integers $N_0, N_1 \in \mathcal{H}_p$ with $N_0 < N_1$ and $N_0 + N_1 = 2p + 2$:*

```
1. If $p \le 11$, return true if $\#E \in \{N_0, N_1\}$ and false otherwise.
2. Set $E_0 \leftarrow E$, $E_1 \leftarrow \tilde{E}$, $m_0 \leftarrow 1$, $m_1 \leftarrow 1$, and $s \leftarrow 0$.
3. Select a random point $P \in E_s(\mathbb{F}_p)$.
4. Use FastOrder to compute the order $n_s$ of the point $Q = m_s P$, assuming
   $n_s$ divides $N_s/m_s$. If this succeeds, set $m_s \leftarrow m_s n_s$ and proceed to step 5.
   If not, provided that $m_0|N_1$, $m_1|N_0$, and $N_0 < N_1$, swap $N_0$ and $N_1$ and go
   to step 3, but otherwise return false.
5. Set $a_1 \leftarrow 2p+2 \bmod m_1$ and $\mathcal{N} \leftarrow \{m_0 x : x \in \mathbb{Z}\} \cap \{m_1 x + a_1 : x \in \mathbb{Z}\} \cap \mathcal{H}_p$.
   If $\mathcal{N} \subseteq \{N_0, N_1\}$ return true, otherwise set $s \leftarrow 1 - s$ and go to step 3.
```

TestCurveOrder computes integers m_s dividing $\#E_s$ by alternately computing the orders of random points on E and $\tilde{E}$. If an order computation fails (this happens when $n_s \nmid N_s/m_s$), it rules out N_s as a possibility for $\#E$. If both N_0 and N_1 are eliminated, the algorithm returns **false**. Otherwise a divisor n_s of N_s is obtained, and the algorithm continues until it narrows the possibilities for $\#E$ to a nonempty subset of $\{N_0, N_1\}$ (it need not determine which). The set $\mathcal{N}$ computed in step 5 must contain $\#E$, since m_0 divides $\#E$ and m_1 divides $\#\tilde{E}$ (the latter implies $\#E \equiv 2p+2 \bmod m_1$, since $\#E + \#\tilde{E} = 2p+2$). The complexity of the algorithm (and a proof that it terminates) is given by Lemma 6 of Section 7.

A simple implementation of FastOrder appears below, based on a recursive algorithm to compute the order of a generic group element due to Celler and Leedham-Green [16]. By convention, generic groups are written multiplicatively, and we do so here, although we apply FastOrder to the additive groups $E(\mathbb{F}_p)$ and $\tilde{E}(\mathbb{F}_p)$. The function $\omega(N)$ counts the distinct prime factors of N.

Algorithm FastOrder. *Given an element α of a generic group G and a factored integer N, compute the function $\mathcal{A}(\alpha, N)$, defined to be the factored integer $M = |\alpha|$ when M divides N, and 0 otherwise.*

```
1. If $N$ is a prime power $p^n$, compute $\alpha^{p^i}$ for increasing $i$ until the identity is
   reached (in which case return $p^i$) or until $i = n$ (in which case return 0).
2. Let $N = N_1N_2$ with $N_1$ and $N_2$ coprime and $|\omega(N_1) - \omega(N_2)| \leq 1$.
   Recursively compute $M = \mathcal{A}(\alpha^{N_2}, N_1) \cdot \mathcal{A}(\alpha^{N_1}, N_2)$ and return $M$.
```

This algorithm uses $O(\log N \log\log N)$ multiplications (and identity tests) in G. A slightly faster algorithm [64, Alg. 7.4] is used in the proof of Theorem 1. In practice, the implementation of TestCurveOrder and FastOrder is not critical, since most of the time is actually spent performing the scalar multiplications discussed above (these occur in step 3 of Algorithm 1.1 below).

We now give the complete algorithm to find an element of $\mathrm{Ell}_t(\mathbb{F}_p)$. For reasons discussed in the next section, we exclude the j-invariants 0 and 1728.

Algorithm 1.1. *Given $p \in \mathcal{P}_D$, find $j \in \mathrm{Ell}_t(\mathbb{F}_p) - \{0, 1728\}$.*

```
1. Factor $N_0 = p + 1 - t$ and $N_1 = p + 1 + t$, and choose torsion constraints.
2. Generate a batch of random elliptic curves $E_i/\mathbb{F}_p$ with $j(E_i) \notin \{0, 1728\}$ that
   satisfy these constraints and pick a random point $P_i$ on each curve.
3. For each $i$ with $(p+1)P_i = \pm tP_i$, test whether $\#E_i \in \{N_0, N_1\}$ by calling
   TestCurveOrder, using the factorizations of $N_0$ and $N_1$.
4. If $\#E_i \in \{N_0, N_1\}$ for some $i$, output $j(E_i)$; otherwise return to step 2.
```

The torsion constraints chosen in step 1 may be precomputed by Algorithm 2.1 in the process of selecting $S \subset \mathcal{P}_D$. In step 2 we may generate E_i with m-torsion as described in Section 3.2; as a practical optimization, if $X_1(m)$ has genus 0 we generate both E_i and P_i using the parametrizations in [3]. In step 3 the point P_i can also be used as the first random point chosen in TestCurveOrder. The condition $(p+1)P_i = \pm tP_i$ is tested by performing scalar multiplications in parallel, as described above. When torsion constraints determine the sign of t, we instead test whether $(p+1-t)P_i = 0$ or $(p+1+t)P_i = 0$, as appropriate.

4. Isogeny volcanoes

The previous section addressed the first step in computing $H_D \bmod p$: finding an element of $\mathrm{Ell}_t(\mathbb{F}_p)$. In this section we address the next two steps: finding an element of $\mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$ and enumerating $\mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$. This yields the roots of $H_D \bmod p$.

We utilize the graph of ℓ-isogenies defined on $\mathrm{Ell}_t(\mathbb{F}_p)$. We regard this as an undirected graph, noting that the dual isogeny [61, §III.6] lets us traverse edges in either direction. We permit self-loops in our graphs, but not multiple edges.

Definition 1. Let ℓ be prime. An *ℓ-volcano* is an undirected graph with vertices partitioned into levels $V_0, \dots, V_d$, in which the subgraph on V_0 (the *surface*) is a regular connected graph of degree at most 2, and also:

1. For $i > 0$, each vertex in V_i has exactly one edge leading to a vertex in V_{i-1}, and every edge not on the surface is of this form.
2. For $i < d$, each vertex in V_i has degree $\ell + 1$.

The surface V_0 of an ℓ-volcano is either a single vertex (possibly with a self-loop), two vertices connected by an edge, or a (simple) cycle on more than two vertices, which is the typical case. We call V_d the *floor* of the volcano, which coincides with the surface when $d = 0$. For $d > 0$ the vertices on the floor have degree 1, and in every case their degree is at most 2; all other vertices have degree $\ell + 1 > 2$.

We refer to d as the *depth* of the ℓ-volcano. The term “height” is also used [54], but “depth” better suits our indexing of the levels V_i and is consistent with [46].

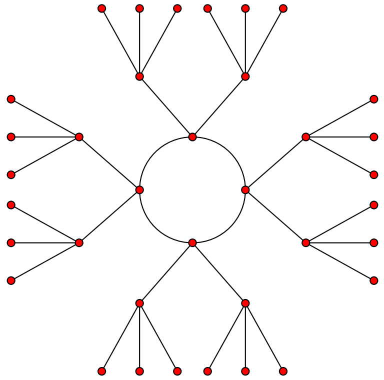

Figure 1. A 3-volcano of depth 2, with a 4-cycle on the surface.

Definition 2. For a prime $\ell \neq p$, let $\Gamma_{\ell,t}(\mathbb{F}_p)$ be the undirected graph with vertex set $\mathrm{Ell}_t(\mathbb{F}_p)$ that contains the edge (j_1, j_2) if and only if $\Phi_\ell(j_1, j_2) = 0$.

Here Φ_ℓ denotes the classical modular polynomial. With at most two exceptions, the components of $\Gamma_{\ell,t}(\mathbb{F}_p)$ are ℓ-volcanoes. The level at which $j(E) \in \mathrm{Ell}_t(\mathbb{F}_p)$ resides in its ℓ-volcano is determined by the power of ℓ dividing the conductor of $\mathrm{End}(E)$.

The discriminant D may be written as $D = u^2 D_K$, where D_K is the discriminant of the maximal order $\mathcal{O}_K$ containing $\mathcal{O}$ and $u = [\mathcal{O}_K : \mathcal{O}]$ is the conductor of $\mathcal{O}$. We also have the discriminant

$$D_\pi = t^2 - 4p = v^2 D = w^2 D_K \tag{12}$$

of the order $\mathbb{Z}[\pi] \subseteq \mathcal{O}_K$ with conductor $w = uv$, generated by the Frobenius endomorphism π with trace t (note $\pi = \pi_E$ for all $j(E) \in \mathrm{Ell}_t(\mathbb{F}_p)$). The order $\mathcal{O}$ contains $\mathbb{Z}[\pi]$, and for any $j(E) \in \mathrm{Ell}_t(\mathbb{F}_p)$ we have $\mathbb{Z}[\pi] \subseteq \mathrm{End}(E) \subseteq \mathcal{O}_K$. Curves with $\mathrm{End}(E) \cong \mathbb{Z}[\pi]$ lie on the floor of their ℓ-volcano, while those with $\mathrm{End}(E) \cong \mathcal{O}_K$ lie on the surface. More generally, the following proposition holds.

Proposition 2. *Let $p \in \mathcal{P}_D$ and let $\ell \neq p$ be a prime. The components of $\Gamma_{\ell,t}(\mathbb{F}_p)$ that do not contain $j = 0, 1728$ are ℓ-volcanoes of depth $d = \nu_\ell(w)$. Each has an associated order $\mathcal{O}_0$, with $\mathbb{Z}[\pi] \subseteq \mathcal{O}_0 \subseteq \mathcal{O}_K$ and $\ell \nmid [\mathcal{O}_K : \mathcal{O}_0]$, and we have*

$$j(E) \in V_i \quad \Longleftrightarrow \quad \operatorname{End}(E) \cong \mathcal{O}_i,$$

where $\mathcal{O}_i$ is the order of index ℓ^i in $\mathcal{O}_0$.

Here ν_ℓ denotes the ℓ-adic valuation (so $\ell^d | w$ but $\ell^{d+1} \nmid w$). The proposition follows essentially from [46, Prop. 23]. See [29, Lemmas 2.1-6] for additional details and [71, Thm. 1.19, Prop. 12.20] for properties of Φ_ℓ.

We have excluded $j = 0, 1728$ (which can arise only when $D_K = -3, -4$) for technical reasons; see [71, Rem. 12.21]. However, a nearly equivalent statement holds; only the degrees of the vertices 0 and 1728 are affected.

4.1. **Obtaining an element of** $\operatorname{Ell}_{\mathcal{O}}(\mathbb{F}_p)$**.** Given $j(E) \in \operatorname{Ell}_t(\mathbb{F}_p) - \{0, 1728\}$, we may apply Proposition 2 to obtain an element of $\operatorname{Ell}_{\mathcal{O}}(\mathbb{F}_p)$. Let u and u_E be the conductors of $\mathcal{O}$ and $\operatorname{End}(E)$, respectively; both u and u_E divide w, the conductor of $D_\pi = t^2 - 4p$. Suppose $\nu_\ell(u_E) \neq \nu_\ell(u)$ for some prime ℓ. If we replace $j = j(E)$ by a vertex at level $\nu_\ell(u)$ in j's ℓ-volcano, we then have $\nu_\ell(u_E) = \nu_\ell(u)$. Proposition 2 assures us that this "adjustment" only affects the power of ℓ dividing u_E. Repeating this for each prime $\ell | w$, we eventually have $u_E = u$ and $j(E) \in \operatorname{Ell}_{\mathcal{O}}(\mathbb{F}_p)$.

To change location in an ℓ-volcano we walk a *path*, which we define to be a sequence of vertices $j_0, \ldots, j_n$ connected by edges (j_k, j_{k+1}), such that $j_{k-1} \neq j_{k+1}$ for all $0 < k < n$ (this condition is enforced by never taking a backward step).

Paths in $\Gamma_{\ell,t}(\mathbb{F}_p)$ are computed by choosing an initial edge (j_0, j_1), and for $k > 0$ extending the path $j_0, \ldots, j_k$ by picking a root j_{k+1} of the polynomial

$$f(X) = \Phi_\ell(X, j_k)/(X - j_{k-1})^e \in \mathbb{F}_p[x].$$

Here e is the multiplicity of the root j_{k-1} in $\Phi_\ell(X, j_k)$, equal to one in all but a few special cases (see [29, Lemma 2.6 and Thm. 2.2]). If $f(X)$ has no roots in $\mathbb{F}_p$, then j_k has no neighbors other than j_{k-1} and the path must end at j_k.

When a path has $j_k \in V_i$ and $j_{k+1} \in V_{i+1}$, we say the path *descends* at k. Once a path starts descending, it must continue to do so. If a path descends at every step and terminates at the floor, we call it a *descending path*, as in [29, Def. 4.1].

We now present an algorithm to determine the level of a vertex j in an ℓ-volcano, following Kohel [46, p. 46]. When walking a path, we suppose neighbors are picked uniformly at random whenever there is a choice to be made.

Algorithm FindLevel. *Compute the level of j in an ℓ-volcano of depth d*:

```
1. If $\deg(j) \neq \ell + 1$, then return $d$; otherwise let $j_1 \neq j_2$ be neighbors of $j$.
2. Walk a path of length $k_1 \leq d$ extending $(j, j_1)$.
3. Walk a path of length $k_2 \leq k_1$ extending $(j, j_2)$.
4. Return $d - k_2$.
```

If FindLevel terminates in step 1, then j is on the floor at level d. The paths walked in steps 2 and 3 are extended as far as possible, up to the specified bound. If j is on the surface, then these paths both have length d, and otherwise at least one of them is a descending path of length k_2. In both cases, j is on level $d - k_2$.

We use the algorithms below to change levels in an ℓ-volcano of depth $d > 0$.

Algorithm Descend. *Given $j \in V_k \neq V_d$, return $j' \in V_{k+1}$:*

```
1. If $k = 0$, walk a path $(j = j_0, \dots, j_n)$ to the floor and return $j' = j_{n-d+1}$.
2. Otherwise, let $j_1$ and $j_2$ be distinct neighbors of $j$.
3. Walk a path of length $d - k$ extending $(j, j_1)$ and ending in $j^*$.
4. If $\deg(j^*) = 1$, then return $j' = j_1$; otherwise return $j' = j_2$.
```

Algorithm Ascend. *Given $j \in V_k \neq V_0$, return $j' \in V_{k-1}$:*

```
1. If $\deg(j) = 1$, then let $j'$ be the neighbor of $j$ and return $j'$;
   otherwise let $j_1, \dots, j_{\ell+1}$ be the neighbors of $j$.
2. For each $i$ from 1 to $\ell$:
   a. Walk a path of length $d - k$ extending $(j, j_i)$ and ending in $j^*$.
   b. If $\deg(j^*) > 1$, then return $j' = j_i$.
3. Return $j' = j_{\ell+1}$.
```

The correctness of Descend and Ascend is easily verified. We note that if $k = 0$ in Descend, then the expected value of n is at most $d + 2$ (for any ℓ).

We now give the algorithm to find an element $j' \in \mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$, given $j \in \mathrm{Ell}_t(\mathbb{F}_p)$. We use a bound L on the primes $\ell | w$, reverting to a computation of the endomorphism ring to address $\ell > L$, as discussed below. This is never necessary when D is fundamental, but it may arise when the conductor of D has a large prime factor.

Algorithm 1.2. *Let $p \in \mathcal{P}_D$, let u be the conductor of D, and let $w = uv$, where $v = v(p)$. Given $j \in \mathrm{Ell}_t(\mathbb{F}_p) - \{0, 1728\}$, find $j' \in \mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$:*

```
1. For each prime $\ell | w$ with $\ell \leq L = \max(\log |D|, v)$:
   a. Use FindLevel to determine the level of $j$ in its $\ell$-volcano.
   b. Use Descend and Ascend to obtain $j'$ at level $\nu_\ell(u)$ and set $j \leftarrow j'$.
2. If $u$ is not $L$-smooth, verify that $j \in \mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$ and abort if not.
3. Return $j' = j$.
```

The verification in step 2 involves computing $\mathrm{End}(E)$ for an elliptic curve $E/\mathbb{F}_p$ with $j(E) = j$. Here we may use the algorithm in [10], or Kohel's algorithm [46]. The former is faster in practice (with a heuristically subexponential running time), but for the proof of Theorem 1 we use the $O(p^{1/3})$ complexity bound of Kohel's algorithm, which depends only on the GRH.

For $p \in S$, we expect v to be small, $O(\log^{3+\epsilon} |D|)$ under the GRH, and heuristically $O(\log^{1/2} |D|)$. Provided u does not contain a prime larger than L, the running time of Algorithm 1.2 is polynomial in $\log |D|$, under the GRH.

However, if u is divisible by a prime $\ell > L$, we want to avoid the cost of computing ℓ-isogenies. Such an ℓ cannot divide v (since $L \geq v$), so our desired j' must lie on the floor of its ℓ-volcano. When ℓ is large, it is highly probable that our initial j is already on the floor (this is where most of the vertices in an ℓ-volcano lie), and this will still hold in step 2. Since $L \geq \log |D|$ is asymptotically larger than the number of prime factors of u, the probability of a failure in step 2 is $o(1)$. If Algorithm 1.2 aborts, we call Algorithm 1.1 again and retry.

If D_K is -3 or -4, then j may lie in a component of $\Gamma_{\ell,t}(\mathbb{F}_p)$ containing 0 or 1728. However, provided we never pick 0 or 1728 when choosing a neighbor, FindLevel, Descend, and Ascend will correctly handle this case.

4.2. **Enumerating** $\mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_{\mathbf{p}})$**.** Having obtained $j_0 \in \mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$, we now wish to enumerate the rest of $\mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$. We assume $h(D) > 1$ and apply the group action of $\mathrm{cl}(D)$ to the set $\mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$. Let ℓ be a prime not dividing the conductor u of D with $(\frac{D}{\ell}) \neq -1$. Then ℓ can be uniquely factored in $\mathcal{O}$ into conjugate prime ideals as $(\ell) = \mathfrak{a}\bar{\mathfrak{a}}$, where $\mathfrak{a}$ and $\bar{\mathfrak{a}}$ both have norm ℓ. The ideals $\mathfrak{a}$ and $\bar{\mathfrak{a}}$ are distinct when $(\frac{D}{\ell}) = 1$, and in any case the ideal classes $[\mathfrak{a}]$ and $[\bar{\mathfrak{a}}]$ are inverses. The orders of $[\mathfrak{a}]$ and $[\bar{\mathfrak{a}}]$ in $\mathrm{cl}(D)$ are equal, and we denote their common value by $\mathrm{ord}_D(\ell)$. The following proposition follows immediately from Propositions 1 and 2.

Proposition 3. *Let $\ell \neq p$ be a prime such that $\ell \nmid u$ and $(\frac{D}{\ell}) \neq -1$. Then every element of $\mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$ lies on the surface V_0 of its ℓ-volcano, and $\#V_0 = \mathrm{ord}_D(\ell)$.*

If $\mathrm{ord}_D(\ell) = h(D)$, then $\mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$ is equal to the surface of the ℓ-volcano containing j_0, but in general we must traverse several volcanoes to enumerate $\mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$. We first describe how to walk a path along the surface of a single ℓ-volcano.

When ℓ does not divide v, every ℓ-volcano in $\Gamma_{\ell,t}(\mathbb{F}_p)$ has depth zero. In this case walking a path on the surface is trivial: for $\#V_0 > 2$ we choose one of the two roots of $\Phi_\ell(X, j_0)$, and every subsequent step is determined by the single root of the polynomial $f(X) = \Phi_\ell(X, j_i)/(X - j_{i-1})$. The cost of each step is then

$$O(\ell^2 + \mathsf{M}(\ell)\log p) \tag{13}$$

operations in $\mathbb{F}_p$, where $\mathsf{M}(n)$ is the complexity of multiplication (the first term is the time to evaluate $\Phi_\ell(X, j_i)$; the second term is the time to compute $X^p \bmod f$).

While it is simpler to restrict ourselves to primes $\ell \nmid v$ (there are infinitely many ℓ we might use), as a practical matter, the time spent enumerating $\mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$ depends critically on ℓ. Consider $\ell = 2$ versus $\ell = 7$. The cost of finding a root of $f(X)$ when f has degree 7 may be 10 or 20 times the cost when f has degree 2. We much prefer $\ell = 2$, even when the 2-volcano has depth $d > 0$ (necessarily the case when $(\frac{D}{2}) = 1$). The following algorithm allows us to handle ℓ-volcanoes of any depth.

Algorithm WalkSurfacePath. *Given $j_0 \in V_0$ in an ℓ-volcano of depth d and a positive integer $n < \#V_0$, return a path $j_0, j_1 \ldots, j_n$ contained in V_0:*

```
1. If $\deg(j_0) = 1$, then return the path $j_0, j_1$, where $j_1$ is the neighbor of $j_0$.
   Otherwise, walk a path $j_0, \ldots, j_d$ and set $i \leftarrow 0$.
2. While $\deg(j_{i+d}) = 1$, replace $j_{i+1}, \ldots, j_{i+d}$ by extending the path $j_0, \ldots, j_i$
   by $d$ steps, starting from a random unvisited neighbor $j'_{i+1}$ of $j_i$.
3. Extend the path $j_0, \ldots, j_{i+d}$ to $j_0, \ldots, j_{i+d+1}$, then set $i \leftarrow i + 1$.
4. If $i = n$, then return $j_0, \ldots, j_n$, otherwise go to step 2.
```

When $d = 0$ the algorithm necessarily returns a path that is contained in V_0. Otherwise, the path extending $d+1$ steps beyond $j_i \in V_0$ in step 3 guarantees that $j_{i+1} \in V_0$. The algorithm maintains (for the current value of i) a list of visited neighbors of j_i to facilitate the choice of an unvisited neighbor in step 2.

To bound the expected running time, we count the vertices examined during its execution, that is, the number of vertices whose neighbors are computed.

Proposition 4. *Let the random variable X be the number of vertices examined by* WalkSurfacePath. *If $\#V_0 = 2$, then $\mathbf{E}[X] = d + 1 + ld/2$, and otherwise*

$$\mathbf{E}[X] \leq d + (1 + (\ell - 1)d/2)n.$$

Proof. If $d = 0$, then WalkSurfacePath examines exactly n vertices and the proposition holds, so we assume $d > 0$ and note that $\deg(j_0) > 1$ in this case. We partition the execution of the algorithm into phases, with phase -1 consisting of step 1, and the remaining phases corresponding to the value of i. At the start of phase $i \geq 0$ we have $j_i \in V_0$ and the path $j_0, \ldots, j_{i+d}$. Let the random variable X_i be the number of vertices examined in phase i, so that $X = X_{-1} + X_0 + \cdots + X_n$. We have $X_{-1} = d$ and $X_n = 0$. For $0 \leq i < n$ we have $X_i = 1 + md$, where m counts the number of incorrect choices of j_{i+1} (those not in V_0).

We first suppose $\#V_0 = 2$. In this case exactly one of the $\ell + 1$ neighbors of j_0 lies in V_0. Conditioning on m we obtain

$$\mathbf{E}[X_0] = \sum_{m=0}^{\ell} \Bigl(1 + md\Bigr) \frac{1}{\ell + 1 - m} \prod_{k=0}^{m-1} \left(\frac{\ell - k}{\ell + 1 - k}\right) = \sum_{m=0}^{\ell} \frac{1 + md}{\ell + 1} = 1 + ld/2.$$

This yields

$$\mathbf{E}[X] = \mathbf{E}[X_{-1}] + \mathbf{E}[X_0] + \mathbf{E}[X_1] = d + 1 + ld/2,$$

as desired. We now assume $\#V_0 > 2$. Then two of j_0's neighbors lie in V_0, and we find that $\mathbf{E}[X_0] = 1 + (\ell - 1)d/3$. For $i > 1$ we exclude the neighbor j_{i-1} of j_i and obtain $\mathbf{E}[X_i] = 1 + (\ell - 1)d/2$. Summing expectations completes the proof. □

Using an estimate of the time to find the roots of a polynomial of degree ℓ in $\mathbb{F}_p[X]$, we may apply Proposition 4 to optimize the choice of the primes ℓ that we use when enumerating $\mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$, as discussed in the next section. As an example, if $(\frac{D}{2}) = 1$ and $\nu_2(v) = 2$, then we need to solve an average of roughly 2 quadratic equations for each vertex when we walk a path along the surface of a 2-volcano in $\Gamma_{\ell,t}(\mathbb{F}_p)$. This is preferable to using any $\ell > 2$, even when $\ell \nmid v$. On the other hand, if $(\frac{D}{5}) = (\frac{D}{7}) = 1$ and $5|v$ but $7 \nmid v$, we likely prefer $\ell = 7$ to $\ell = 5$.

We now present Algorithm 1.3, which, given $j_0 \in \mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$ and suitable lists of primes ℓ_i and integers r_i, outputs the elements of $\mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p) - \{j_0\}$. It may be viewed as a generalization of WalkSurfacePath to k dimensions.

Algorithm 1.3. *Given $j_0 \in \mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$, primes $\ell_1, \ldots, \ell_k$ with $\ell_i \nmid u$ and $(\frac{D}{\ell_i}) \neq -1$, and integers $r_1, \ldots, r_k$ with $1 < r_i \leq \mathrm{ord}_D(\ell_i)$:*

1. *Use* WalkSurfacePath *to compute a path $j_0, j_1, \ldots, j_{r_k - 1}$ of length $r_k - 1$ on the surface of the ℓ_k-volcano containing j_0, and output $j_1, \ldots, j_{r_k-1}$.*
2. *If $k > 1$, then for i from 0 to $r_k - 1$ recursively call Algorithm* 1.3 *using j_i, the primes $\ell_1, \ldots, \ell_{k-1}$, and the integers $r_1, \ldots, r_{k-1}$.*

Proposition 2 implies that Algorithm 1.3 outputs a subset of $\mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$, since $j_0, j_1, \ldots, j_{r_k-1}$ all lie on the surface of the same ℓ_k-volcano (and this applies recursively). To ensure that Algorithm 1.3 outputs all the elements of $\mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p) - \{j_0\}$, we use a polycyclic presentation for $\mathrm{cl}(D)$, as defined in the next section.

5. Polycyclic presentations of finite abelian groups

To obtain suitable sequences $\ell_1, \ldots, \ell_k$ and $r_1 \ldots, r_k$ for use with Algorithm 1.3, we apply the theory of polycyclic presentations [37, Ch. 8]. Of course $\mathrm{cl}(D)$ is a finite abelian group, but the concepts we need have been fully developed in the setting of polycyclic groups and conveniently specialize to the finite abelian case.

Let $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_k)$ be a sequence of generators for a finite abelian group G, and let $G_i = \langle \alpha_1, \dots, \alpha_i \rangle$ be the subgroup generated by $\alpha_1, \dots, \alpha_i$. The series

$$1 = G_0 \leq G_1 \leq \cdots \leq G_{k-1} \leq G_k = G$$

is necessarily a *polycyclic series*, that is, a subnormal series in which each quotient G_i/G_{i-1} is a cyclic group. Indeed, $G_i/G_{i-1} = \langle \alpha_i G_{i-1} \rangle$, and $\boldsymbol{\alpha}$ is a *polycyclic sequence* for G. We say that $\boldsymbol{\alpha}$ is *minimal* if none of the quotients are trivial.

When $G = \prod \langle \alpha_i \rangle$, we have $G_i/G_{i-1} \cong \langle \alpha_i \rangle$ and call $\boldsymbol{\alpha}$ a *basis* for G, but this is a special case. For abelian groups, G_i/G_{i-1} is isomorphic to a subgroup of $\langle \alpha_i \rangle$, but it may be a proper subgroup, even when $\boldsymbol{\alpha}$ is minimal.

The sequence $r(\boldsymbol{\alpha}) = (r_1, \dots, r_k)$ of *relative orders* for $\boldsymbol{\alpha}$ is defined by

$$r_i = |G_i : G_{i-1}|.$$

We necessarily have $\prod r_i = |G|$, and if $\boldsymbol{\alpha}$ is minimal, then $r_i > 1$ for all i. The sequences $\boldsymbol{\alpha}$ and $r(\boldsymbol{\alpha})$ allow us to uniquely represent every $\beta \in G$ in the form

$$\beta = \boldsymbol{\alpha}^{\boldsymbol{x}} = \alpha_1^{x_1} \cdots \alpha_k^{x_k}.$$

Lemma 1. *Let $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_k)$ be a sequence of generators for a finite abelian group G, let $r(\boldsymbol{\alpha}) = (r_1, \dots, r_k)$, and let $X(\boldsymbol{\alpha}) = \{\boldsymbol{x} \in \mathbb{Z}^k : 0 \leq x_i < r_i\}$.*

1. *For each $\beta \in G$ there is a unique $\boldsymbol{x} \in X(\boldsymbol{\alpha})$ such that $\beta = \boldsymbol{\alpha}^{\boldsymbol{x}}$.*
2. *The vector $\boldsymbol{x}$ such that $\alpha_i^{r_i} = \boldsymbol{\alpha}^{\boldsymbol{x}}$ has $x_j = 0$ for $j \geq i$.*

Proof. See Lemmas 8.3 and 8.6 in [37]. □

The vector $\boldsymbol{x}$ is the *discrete logarithm* (exponent vector) of β with respect to $\boldsymbol{\alpha}$. The relations $\alpha_i^{r_i} = \boldsymbol{\alpha}^{\boldsymbol{x}}$ are called *power relations* and may be used to define a (consistent) polycyclic presentation for an abelian group G, as in [37, Def. 8.7].

We now show that a minimal polycyclic sequence for $\mathrm{cl}(D)$ provides suitable inputs for Algorithm 1.3.

Proposition 5. *Let $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_k)$ be a minimal polycyclic sequence for* $\mathrm{cl}(D)$ *with relative orders $r(\boldsymbol{\alpha}) = (r_1, \dots, r_k)$, and let $\ell_1, \dots, \ell_k$ be primes for which α_i contains an invertible ideal of norm ℓ_i. Given $j_0 \in \mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$, the primes ℓ_i, and the integers r_i, Algorithm* 1.3 *outputs each element of* $\mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p) - \{j_0\}$ *exactly once.*

Proof. As previously noted, Proposition 2 implies that the outputs of Algorithm 1.3 are elements of $\mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$. Since $\prod r_i = \#\,\mathrm{cl}(D) = \#\,\mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$, by Proposition 1, we need only show that the outputs are distinct (and not equal to j_0).

To each vertex of the isogeny graph output by Algorithm 1.3 we associate a vector $\boldsymbol{x} \in X(\boldsymbol{\alpha})$ that identifies its position relative to j_0 in the sequence of paths computed. The vector $(x_1, \dots, x_k)$ identifies the vertex reached from j_0 via a path of length x_k on the surface of the ℓ_k-volcano, followed by a path of length x_{k-1} on the surface of the ℓ_{k-1}-volcano, and so forth. We associate the zero vector to j_0.

Propositions 1 and 2 imply that the vector $\boldsymbol{x} = (x_1, \dots, x_k)$ corresponds to the action of some $\beta_{\boldsymbol{x}} \in \mathrm{cl}(D)$. For each integer t_k in the interval $[0, r_k)$, the set of vectors of the form $(*, \dots, *, t_k)$ corresponds to a coset of G_{k-1} in the polycyclic series for $G = \mathrm{cl}(D)$. These cosets are distinct, regardless of the direction chosen by Algorithm 1.3 when starting its path on the ℓ_k-volcano (note that α_k and α_k^{-1} have the same relative order r_k). Proceeding inductively, for each choice of integers $t_i, t_{i+1}, \dots, t_k$ with $t_j \in [0, r_j)$ for $i \leq j \leq k$, the set of vectors of the form $(*, \dots, *, t_i, t_{i+1}, \dots, t_k)$ corresponds to a distinct coset of G_{i-1}, regardless of the direction chosen by Algorithm 1.3 on the surface of the ℓ_i-volcano. Each coset of the cyclic group G_0 corresponds bijectively to a set of vectors of the form $(*, t_j, \ldots, t_k)$. It follows that the $\beta_{\boldsymbol{x}}$'s are all distinct. The action of $\mathrm{cl}(D)$ is faithful, hence the outputs of Algorithm 1.3 are distinct. □

5.1. **Computing an optimal polycyclic presentation.** Let $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_n)$ be a sequence of generators for a finite abelian group G, ordered by increasing cost (according to some cost function). Then $\boldsymbol{\gamma}$ is a polycyclic sequence, and we may compute $r(\boldsymbol{\gamma}) = (r_1, \ldots, r_n)$. If we remove from $\boldsymbol{\gamma}$ each γ_i for which $r_i = 1$ and let $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_k)$ denote the remaining subsequence, then $\boldsymbol{\alpha}$ is a minimal polycyclic sequence for G. We call $\boldsymbol{\alpha}$ the *optimal* polycyclic sequence derived from $\boldsymbol{\gamma}$. It has $\alpha_1 = \gamma_1$ with minimal cost, and for $i > 1$ each α_i is the least-cost element not already contained in $G_{i-1} = \langle \alpha_1, \ldots, \alpha_{i-1} \rangle$.

We now give a generic algorithm to compute $r(\boldsymbol{\gamma})$ and a vector $s(\boldsymbol{\gamma})$ that encodes the power relations. From $r(\boldsymbol{\gamma})$ and $s(\boldsymbol{\gamma})$, we can easily derive $\boldsymbol{\alpha}, r(\boldsymbol{\alpha})$, and $s(\boldsymbol{\alpha})$. We define $s(\boldsymbol{\gamma})$ using a bijection $X(\boldsymbol{\gamma}) \to \{z \in \mathbb{Z} : 0 \le z < |G|\}$ given by

$$Z(\boldsymbol{x}) = \sum_{1 \le j \le n} N_j x_j, \qquad \text{where} \quad N_j = \prod_{1 \le i < j} r_i. \tag{14}$$

For each power relation $\gamma_i^{r_i} = \boldsymbol{\gamma}^{\boldsymbol{x}}$, we set $s_i = Z(\boldsymbol{x})$. The formula

$$x_j = \lfloor s_i / N_j \rfloor \bmod r_j \tag{15}$$

recovers the component x_j of the vector $\boldsymbol{x}$ for which $s_i = Z(\boldsymbol{x})$.

Algorithm 2.2. *Given $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_n)$ generating a finite abelian group G:*

```
1. Let $T$ be an empty table and call TableInsert($T, 1_G$) (so $T[0] = 1_G$).
2. For $i$ from 1 to $n$:
3.     Set $\beta \leftarrow \gamma_i$, $r_i \leftarrow 1$, and $N \leftarrow$ TableSize($T$).
4.     Until $s_i \leftarrow$ TableLookup($T, \beta$) succeeds:
5.         For $j$ from 0 to $N-1$: TableInsert($T, \beta \cdot T[j]$).
6.         Set $\beta \leftarrow \beta\gamma_i$ and $r_i \leftarrow r_i + 1$.
7. Output $r(\boldsymbol{\gamma}) = (r_1, \ldots, r_n)$ and $s(\boldsymbol{\gamma}) = (s_1, \ldots, s_n)$.
```

The table T stores elements of G in an array, placing each inserted element in the next available entry. The function TableLookup(T, β) returns an integer j for which $T[j] = \beta$ or fails if no such j exists (when j exists it is unique). In practice lookups are supported by an auxiliary data structure, such as a hash table, maintained by TableInsert. When group elements are uniquely identified, as with $\mathrm{cl}(D)$, the cost of table operations is typically negligible.

Proposition 6. *Algorithm* 2.2 *is correct. It uses $|G|$ nontrivial group operations, makes $|G|$ calls to* TableInsert, *and makes $\sum r_i$ calls to* TableLookup.

Proof. We will prove inductively that $T[Z(\boldsymbol{x})] = \boldsymbol{\gamma}^{\boldsymbol{x}}$ and that each time the loop in step 4 terminates, the values of r_i and s_i are correct and T holds G_i.

When $i = 1$ the algorithm computes $T[r_1] = \gamma_1^{r_1} T[0]$ for $r_1 = 1, 2, \ldots$, until $\gamma_1^{r_1} = T[0] = 1$, at which point $r_1 = |\gamma_i|$, $s_1 = 0$, and T holds G_1, as desired.

For $i > 1$ we have $N = N_{i-1}$ and T holds G_{i-1} with $T[Z(\boldsymbol{x})] = \boldsymbol{\gamma}^{\boldsymbol{x}}$, by the inductive hypothesis. For $r_i = 1, 2, \ldots$, if $\beta = \gamma_i^{r_i}$ is not in T, the algorithm computes $T[r_i N + j] = \gamma_i^{r_i} T[j]$, for $0 \le j < N$, placing the coset $\gamma_i^{r_i} G_{i-1}$ in T.

When it finds $\gamma_i^{r_i} = T[s_i]$, the table T contains all cosets of the form $\gamma_i^{r_i} G_{i-1}$ (since G is abelian), hence T holds G_i. It follows that $r_i = |G_i : G_{i-1}|$ and s_i is correct.

When the algorithm terminates, T holds $G_n = G$, and every element of G is inserted exactly once. A group operation is performed for each call to TableInsert, but in each execution of step 5 the first of these is trivial, and we instead count the nontrivial group operation in step 6. The number of calls to TableLookup is clearly the sum of the r_i, which completes the proof. □

The complexity of Algorithm 2.2 is largely independent of $\boldsymbol{\gamma}$. When $\boldsymbol{\gamma}$ contains every element of G, Algorithm 2.2 is essentially optimal. However, if $\boldsymbol{\gamma}$ has size $n = o(|G|^{1/2})$, we can do asymptotically better with an $O(n|G|^{1/2})$ algorithm. This is achieved by computing a basis $\boldsymbol{\alpha}$ for G via a generic algorithm (as in [14, 64, 66, 67]) and then determining the representation of each $\gamma_i = \boldsymbol{\alpha}^{\boldsymbol{x}}$ in this basis using a vector discrete logarithm algorithm (such as [64, Alg. 9.3]). It is then straightforward to compute $|G_i|$ for each i, and from this we obtain $r_i = |G_i : G_{i-1}|$. The power relations can then be computed using discrete logarithms with respect to $\boldsymbol{\gamma}$. In the specific case $G = \mathrm{cl}(D)$, one may go further and use a nongeneric algorithm to compute a basis $\boldsymbol{\alpha}$ in subexponential time (under the ERH) [34] and apply a vector form of the discrete logarithm algorithm in [69].

5.2. **Application to D.** For the practical range of D, the group $G = \mathrm{cl}(D)$ is relatively small (typically $|G| < 10^8$), and the constant factors make Algorithm 2.2 faster than alternative approaches; even in the largest examples of Section 8 it takes only a few seconds. Asymptotically, Algorithm 2.2 uses $O(|D|^{1/2+\epsilon})$ time and $O(|D|^{1/2}\log^2|D|)$ space to compute an optimal polycyclic sequence for $\mathrm{cl}(D)$. In fact, under the GRH, we can compute a separate polycyclic sequence for every $v(p)$ arising among the primes $p \in S$ that are selected by Algorithm 2.1 (Section 3.3) within the same complexity bound, by Lemma 3 (Section 7).

We uniquely represent elements of $\mathrm{cl}(D)$ with primitive, reduced, binary quadratic forms $ax^2 + bxy + cy^2$, where a corresponds to the norm of a reduced ideal representing its class. For the sequence $\boldsymbol{\gamma}$ we use forms with $a = \ell$ prime, constructed as in [15, Alg. 3.3]. Under the ERH, restricting to $\ell \le 6\log^2|D|$ yields a sequence of generators for $\mathrm{cl}(D)$, by [4]. To obtain an unconditional result, we precompute $h(D)$ and extend $\boldsymbol{\gamma}$ dynamically until Algorithm 2.2 reaches $N = h(D)$.

We initially order the elements γ_i of $\boldsymbol{\gamma}$ by their norm ℓ_i, assuming that this reflects the cost of using the action of γ_i to enumerate $\mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$ via Algorithm 1.3 (Section 4.2). However, for those ℓ_i that divide $v(p)$ we may wish to adjust the relative position of γ_i, since walking the surface of an ℓ_i-volcano with nonzero depth increases the average cost per step. We use Proposition 4 to estimate this cost, which may or may not cause us to change the position of γ_i in $\boldsymbol{\gamma}$. In practice just a few (perhaps one) distinct orderings suffice to optimally address every $v(p)$.

Note that we need not consider the relative orders r_i when ordering $\boldsymbol{\gamma}$. If i is less than j, then Algorithm 1.3 always takes at least as many steps using ℓ_i as it does using ℓ_j. Indeed, the running time of Algorithm 1.3 is typically determined by the choice of α_1: at least half of the steps will be taken on the surface of an ℓ_1-volcano, and if $(\frac{D}{\ell_1}) = 1$, almost all of them will (heuristically).

5.3. **Why not use a basis?** Using a basis to enumerate $\mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$ is rarely optimal, and in the worst case it can be a very poor choice. The ERH does imply that $\mathrm{cl}(D)$

is generated by the classes of ideals with prime norm $\ell \leq 6 \log^2 |D|$, but this set of generators need not contain a basis. As a typical counterexample, consider

$$D_1 = -10007 \cdot 10009 \cdot 10037,$$

the product of the first three primes greater than 10000. The class group has order $h(D_1) = 2^2 \cdot 44029$, where 44029 is prime, and its 2-Sylow subgroup H is isomorphic to $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z}$. Every basis for $\mathrm{cl}(D_1)$ must contain a nontrivial element of H, and these classes have reduced representatives with norms 10007, 10009, and 10037, all of which are greater than $6 \log^2 |D_1| \approx 4583$.

By comparison, Algorithm 2.2 computes an optimal polycyclic sequence for $\mathrm{cl}(D_1)$ with $\ell_1 = 5$ and $\ell_2 = 37$ (and relative orders $r_1 = 88058$ and $r_2 = 2$).

6. Chinese remaindering

As described in Section 2, for each coefficient c of the Hilbert class polynomial we may derive the value of $c \bmod P$ (for any positive integer P) from the values $c_i \equiv c \bmod p_i$ appearing in $H_D \bmod p_i$ (for $p_i \in S$), using an explicit form of the Chinese Remainder Theorem (CRT). We apply

$$c \equiv \sum c_i a_i M_i - rM \bmod P, \tag{6}$$

where $M = \prod p_i$, $M_i = M/p_i$, $a_i = M_i^{-1} \bmod p_i$, and r is the closest integer to $s = \sum c_i a_i / p_i$. Recall that $S \subset \mathcal{P}_D$ is chosen so that $M > 4B$, where B bounds the coefficients of H_D, via Lemma 8. It suffices to approximate each term in the sum s to within $1/(4n)$, where $n = \#S$. If p_{M} denotes the largest p_i, we need $O\big(\log(n(p_{\mathrm{M}} + \log n))\big) = O(\log p_{\mathrm{M}})$ bits of precision to compute r.

To minimize the space required, we accumulate $C = \sum c_i a_i M_i \bmod P$ and an approximation of s as the c_i are computed. This uses $O(\log P + \log p_{\mathrm{M}})$ space per coefficient. We have $h(D)$ coefficients to compute, yielding

$$O\big(h(D)(\log P + \log p_{\mathrm{M}})\big) \tag{16}$$

as our desired space bound.

To achieve this goal without increasing the time complexity of our algorithm, we consider two cases: one in which P is small, which we take to mean

$$\log P \leq \mu \log^3 |D|, \tag{17}$$

for some absolute constant μ, and another in which P is large (not small). The former case is typical when applying the CM method; P may be a cryptographic-size prime, but it is not unreasonably large. The latter case most often arises when we actually want to compute H_D over $\mathbb{Z}$. When $P \geq M$ there is no need to use the explicit CRT, and we apply a standard CRT computation. To treat the intermediate case, where P is large but smaller than M, we use a hybrid approach.

The optimal choice of μ depends on the relative cost of performing $h(D)$ multiplications modulo P versus the cost of computing $H_D \bmod p_i$; we want the former to be small compared to the latter. In practice, the constant factors allow us to make μ quite large, and the intermediate case rarely arises.

6.1. **Fast Chinese remaindering in linear space.** Standard algorithms for fast Chinese remaindering can be found in [70, §10.3]. We apply similar techniques, but use a time/space trade-off to achieve the space bound in (16). These computations involve a *product tree* built from coprime moduli. In our setting these are the primes $p_i \in S$, which we index here as $p_0, \dots, p_{n-1}$.

We define a product tree as a leveled binary tree in which each vertex at level k is either a leaf or the product of its two children at level $k+1$ (we require levels to have an even number of vertices and add a leaf to levels that need one). It is convenient to label the vertices by bit-strings of length k, where the root at level 0 is labeled by the empty string and all other vertices are uniquely labeled by appending the string "0" or "1" to the label of their parent.

Let $d = \lfloor \lg(n-1) \rfloor + 1$ be the number of bits in the positive integer $n-1$. For integers i from 0 to $n-1$, we let $b(i) \in \{0,1\}^d$ denote the bit-string corresponding to the binary representation of i. The products m_x are defined by placing the moduli in leaves as $m_{b(i)} = p_i$, setting $m_x = 1$ for all other leaves, and defining $m_x = m_{x0}m_{x1}$ for all internal vertices.

The modular complements $\overline{m}_x = m/m_x \bmod m_x$ are then obtained by setting $\overline{m}_0 = m_1 \bmod m_0$ and $\overline{m}_1 = m_0 \bmod m_1$, and defining

$$\overline{m}_{x0} = \overline{m}_x m_{x1} \bmod m_{x0} \qquad \text{and} \qquad \overline{m}_{x1} = \overline{m}_x m_{x0} \bmod m_{x1}.$$

In terms of $M_i = M/p_i$, we then have $m = M$ and $\overline{m}_{b(i)} = M_i \bmod p_i$.

Let I_k denote the labels at level k, for $1 \le k \le d$ (and otherwise I_k is empty). One way to compute $\overline{m}_d$ is as follows:

```
1. For $k$ from $d$ to 1, compute $m_x$ for $x \in I_k$.
2. For $k$ from 1 to $d$, compute $\overline{m}_x$ for $x \in I_k$.
```

This uses $O(\mathsf{M}(\log M)\log n)$ and $O(\log M \log n)$ space. Alternatively:

```
1. For $k$ from 1 to d:
2.     For $j$ from $d$ to $k$, compute $m_x$ for $x \in I_j$ (discard $m_y$ for $y \in I_{j+1}$).
3.     Compute $\overline{m}_x$ for $x \in I_k$ (discard $m_y$ for $y \in I_k$ and $\overline{m}_z$ for $z \in I_{k-1}$).
```

This uses $O(\mathsf{M}(\log M)\log^2 n)$ time and $O(\log M)$ space. In general, storing $\lceil \log^\omega n \rceil$ levels uses $O(\mathsf{M}(\log M)\log^{2-\omega} n)$ time and $O(\log M \log^\omega n)$ space, for any postive real $\omega \le 1$.

6.2. **Applying the explicit CRT when P is small.** Assume $\log P \le \mu \log^3 |D|$. We index the set $S \subset \mathcal{P}_D$ as $S = \{p_0, \dots, p_{n-1}\}$ and let $M = \prod p_i$ and $M_i = M/p_i$. As above, we define products m_x and modular complements $\overline{m}_x = m/m_x \bmod m_x$, and similarly define modular complements $\overline{m}'_x = m/m_x \bmod P$.

Algorithm 2.3 (precompute). *Given $S = \{p_0, \dots, p_{n-1}\}$ and P:*

```
1. Compute $\overline{m}_x$ and $\overline{m}'_x$. Save $M \bmod P$.
2. Use $\overline{m}_{b(i)} \equiv M_i \bmod p_i$ to set $a_i \leftarrow M_i^{-1} \bmod p_i$.
3. Use $\overline{m}'_{b(i)} \equiv M_i \bmod P$ to set $d_i \leftarrow a_i M_i \bmod P$.
4. Set $C_j \leftarrow 0$ and $s_j \leftarrow 0$ for $j$ from 0 to $h(D)$.
```

Using the time/space trade-off described above, Algorithm 2.3 has a running time of $O(\mathsf{M}(\log M)\log^2 n)$, using $O(\log M + n\log P)$ space.

We now set $\delta = \lceil \lg n \rceil + 2$, which determines the precision of the integer $s_j \approx 2^\delta r$ that we use to approximate the rational number r in (6).

Algorithm 2.4 (update). *Given $H_D \bmod p_i$ with coefficients c_j:*

```
1. For $j$ from $0$ to $h(D)$:
2.     Set $C_j \leftarrow C_j + c_j d_i \bmod P$.
3.     Set $s_j \leftarrow s_j + \lfloor 2^\delta c_j a_i / p_i \rfloor$.
```

The total running time of Algorithm 2.4 over all $p_i \in S$ may be bounded by

$$O\big(nh(D)\mathsf{M}(\log P) + h(D)\mathsf{M}(\log M + n\log n)\big). \tag{18}$$

Typically the first term dominates, and it is here that we need $\log P = O(\log^3 |D|)$. The space complexity is $O(h(D)(\log P + \log p_{\mathrm{M}} + \log n))$.

Algorithm 2.5 (postcompute). *After computing $H_D \bmod p_i$ for all $p_i \in S$:*

```
1. For $j$ from $0$ to $h(D)$:
2.     Set $C_j \leftarrow C_j - \lfloor 3/4 + 2^{-\delta} s_j \rfloor M \bmod P$.
3. Output $H_D \bmod P$ with coefficients $C_j$.
```

Algorithm 2.5 uses $O(h(D)\mathsf{M}(\log P))$ time and $O(h(D)\log P)$ space. The formulas used by Algorithms 2.4 and 2.5 are taken from [8, Thm. 2.2] (also see [7]).

6.3. Applying the CRT when P is large.

When P is larger than M, we simply compute $H_D \in \mathbb{Z}[X]$ using a standard application of the CRT. That is, we compute $H_D \bmod p_i$ for $p_i \in S$, and then apply

$$c \equiv \sum c_i a_i M_i \bmod M \tag{5}$$

to compute each coefficient of H_D using fast Chinese remaindering [70, §10.3]. Since $M > 2B$, this determines $H_D \in \mathbb{Z}[X]$. Its coefficients lie in the interval $(-P/2, P/2)$, so we regard this as effectively computing $H_D \bmod P$. The total time spent applying the CRT is then $O(h(D)\mathsf{M}(\log M)\log n)$, and the space needed to compute (5) is $O(\log M \log n)$, which is easily smaller than the $O(h(D)\log M)$ bound on the size of H_D (so no time/space trade-off is required).

When P is smaller than M but $\log P > \mu \log^3 |D|$, we combine the two CRT approaches. We group the primes $p_0, \ldots, p_{n-1}$ into products $q_0, \ldots, q_{k-1}$ so that $\log q_j \approx \log P$ (or $q_j > \log P$ is prime). We compute $H_D \bmod q_j$ by applying the usual CRT to the coefficients of $H_D \bmod p_i$, after processing all the p_i dividing q_j. If q_j is prime no work is involved, and otherwise this takes $O(\mathsf{M}(\log P)\log n)$ time per coefficient. We then apply the explicit CRT to the coefficients of $H_D \bmod q_j$, as in Section 6.2, discarding the coefficients of $H_D \bmod q_j$ after they have been processed by Algorithm 2.4. This hybrid approach has a time complexity of

$$O(h(D)(\log M/\log P)\mathsf{M}(\log P)\log n) = O(h(D)\mathsf{M}(\log M)\log n), \tag{19}$$

and uses $O\big(h(D)(\log P + \log p_{\mathrm{M}})\big)$ space.

7. Complexity analysis

We now analyze the complexity of Algorithms 1 and 2, proving Theorem 1 through a series of lemmas. To do so, we apply various number-theoretic bounds that depend on some instance of the extended or generalized Riemann hypothesis. We use the generic label "GRH" to identify all statements that depend (directly or indirectly) on one or more of these hypotheses. As noted in the Introduction, the GRH is used only to obtain complexity bounds; the outputs of Algorithms 1 and 2 are unconditionally correct.

Let $\mathsf{M}(n)$ denote the cost of multiplication, as defined in [70, Ch. 8]. We have

$$\mathsf{M}(n) = O(n \log n \operatorname{llog} n), \tag{20}$$

by [57], where $\operatorname{llog}(n)$ denotes $\log\log n$ (and we use $\operatorname{lllog}(n)$ to denote $\log\log\log n$). Here we focus on asymptotic results and apply (20) throughout, noting that the larger computations in Section 8 make extensive use of algorithms that realize this bound. See Section 7.1 for a practical discussion of $\mathsf{M}(n)$.

Let us recall some key parameters. For a discriminant $D < -4$, we define

$$\mathcal{P}_D = \{p > 3 \text{ prime} : 4p = t^2 - v^2 D \text{ for some } t, v \in \mathbb{Z}_{>0}\}, \tag{2}$$

where $t = t(p)$ and $v = v(p)$ are uniquely determined by p. We select a subset

$$S \subseteq S_z = \{p \in \mathcal{P}_D : p/H(-v(p)^2 D) \leq z\}$$

that satisfies $\prod_{p\in S} p > 4B$, where B bounds the absolute values of the coefficients of H_D. We also utilize prime norms $\ell_1, \ldots, \ell_k$ arising in a polycyclic presentation of $\mathrm{cl}(D)$ that is derived from a set of generators.

(**GRH**) For convenient reference, we note the following bounds:

(i) $h = h(D) = O(|D|^{1/2} \operatorname{llog} |D|)$ (see [52]).
(ii) $b = \lg B + 2 = O(|D|^{1/2} \log |D| \operatorname{llog} |D|)$ (Lemma 8).
(iii) $n = \#S = O(|D|^{1/2} \operatorname{llog} |D|)$ (follows from (ii)).
(iv) $\ell_{\mathrm{M}} = \max\{\ell_1, \ldots, \ell_k\} = O(\log^2 |D|)$ (see [4]).
(v) $z = O(|D|^{1/2} \log^3 |D| \operatorname{llog} |D|)$ (Lemma 2).
(vi) $p_{\mathrm{M}} = \max S = O(|D| \log^6 |D| \operatorname{llog}^8 |D|)$ (Lemma 3).
(vii) $v_{\mathrm{M}} = \max\{v(p) : p \in S\} = O(\log^3 |D| \operatorname{llog}^4 |D|)$ (Lemma 3).

The first three parameters have unconditional bounds that are only slightly larger (see [5, §5.1]), but the last four depend critically on either the ERH or GRH. Heuristic bounds are discussed in Section 7.1.

To prove (v) we use an effective form of the Chebotarev density theorem [48]. Recall that $\mathcal{P}_D$ is the set of primes (greater than 3) that split completely in the ring class field $K_{\mathcal{O}}$ of $\mathcal{O}$. For a positive real number x, let $\pi_1(x, K_{\mathcal{O}}/\mathbb{Q})$ count the primes $p \leq x$ that split completely in $K_{\mathcal{O}}$. Equivalently, $\pi_1(x, K_{\mathcal{O}}/\mathbb{Q})$ counts primes whose image in $\mathrm{Gal}(K_{\mathcal{O}}/\mathbb{Q})$ under the Artin map is the identity element [23, Cor. 5.21]. Applying Theorem 1.1 of [48] then yields

$$\left|\pi_1(x, K_{\mathcal{O}}/\mathbb{Q}) - \frac{\mathrm{Li}(x)}{2h(D)}\right| \leq c_1 \left(\frac{x^{1/2} \log\left(|D|^{h(D)} x^{2h(D)}\right)}{2h(D)} + \log(|D|^{h(D)})\right), \tag{21}$$

as in [5, Eq. 3], where the constant c_1 is effectively computable.

Lemma 2 (GRH). *For any real constant c_3 there is an effectively computable constant c_2 such that $z \geq c_2 h(D) \log^3 |D|$ implies $\#S_z \geq c_3 h(D) \log^3 |D|$.*

Proof. Let $h = h(D)$. We apply (21) to $x = c_0 h^2 \log^4 |D|$, with c_0 to be determined. We assume $D < -4$ and $\log c_0 \geq 2$, which implies $\log x < 4 \log c_0 \log |D|$ (using $h < |D|$ and $\log |D| < |D|^{1/2}$), and $\mathrm{Li}(x) > x/\log x$ for all $x \geq 1$. Negating the expression within the absolute value, we obtain from (21) the inequality

$$\pi_1(x, K_{\mathcal{O}}/\mathbb{Q}) \geq \left(\frac{c_0}{8 \log c_0} - 5c_1\sqrt{c_0} \log c_0\right) h \log^3 |D|.$$

Thus given any constant c_4 we may effectively determine $c_0 \ge e^2$ (using c_1) so that

$$\pi_1(x, K_{\mathcal{O}}/\mathbb{Q}) \ \ge\ c_4 h \log^3|D|.$$

For the set R_x of primes in $\mathcal{P}_D$ bounded by x, we have $\#R_x = \pi_1(x, K_{\mathcal{O}}/\mathbb{Q}) - 2$.

Let v_0 be the least integer such that at least half the primes in R_x have $v(p) \le v_0$. There are v_0 positive integers less than or equal to v_0, and any particular value $v(p) \le v_0$ can arise for at most $2\sqrt{x}$ primes $p \in R_x$, since $t(p) < 2\sqrt{p} \le 2\sqrt{x}$. Therefore $2v_0\sqrt{x} \ge \#R_x/2$, and this implies

$$2v_0\sqrt{c_0}h\log^2|D| \ \ge\ (c_4 h\log^3|D| - 2)/2 \ >\ (c_4/2-1)h\log^3|D|.$$

We thus obtain $v_0 > c_5 \log|D|$, where $c_5 = (c_4/2-1)/\sqrt{4c_0}$, and assume $c_4 > 2$.

For primes $p \in R_x$ with $v(p) \ge v_0$, the lower bound in Lemma 9 implies

$$\frac{p}{H(-v(p)^2D)} \ \le\ \frac{p}{v(p)H(-D)} \ \le\ \frac{x}{c_5 h\log|D|} \ =\ (c_0/c_5)h\log^3|D|.$$

If $z \ge c_2 h\log^3|D|$, with $c_2 = c_0/c_5$, then S_z contains at least half the primes in R_x. Setting $c_4 = \max\{2c_3+2, 3\}$ determines c_0, c_5, and c_2, and completes the proof. □

The primes $p \in S_z$ are enumerated by Algorithm 2.1 (Section 3.3), which gradually increases z until $\sum_{p\in S_z}\lg p > 2b$, where $b = \lg B + 2$.

Lemma 3 (GRH)**.** *When Algorithm* 2.1 *terminates, for every prime $p \in S_z$ we have the bounds $p = O(|D|\log^6|D|\operatorname{llog}^8|D|)$ and $v(p) = O(\log^3|D|\operatorname{llog}^4|D|)$.*

Proof. Let $D = u^2D_K$, where u is the conductor of D. The upper bound in Lemma 9, together with the bound (i) on $h(D)$, implies that for a suitable constant c_2 and sufficiently large $|D|$, the bound

$$H(-v^2D) \ \le\ 12uvH(-D_K)\operatorname{llog}^2(uv+4) \ \le\ c_2 v|D|^{1/2}\operatorname{llog}|D|\operatorname{llog}^2(v|D|)$$

holds for all positive integers v.

Lemma 2, together with bounds (i) and (ii), implies that Algorithm 2.1 achieves $\sum_{p\in S_z}\lg p > 2b$ with $z = O(h(D)\log^3|D|) = O(|D|^{1/2}\log^3|D|\operatorname{llog}|D|)$. Thus for a suitable constant c_3 and sufficiently large $|D|$, the bound

$$p \ \le\ zH(-v(p)^2D) \ \le\ c_3 v(p)|D|\log^3|D|\operatorname{llog}^2|D|\operatorname{llog}^2(v(p)|D|) \tag{22}$$

holds for all $p \in S_z$. We also have $v(p) \le 2\sqrt{p/|D|}$, since $4p = t(p)^2 - v(p)^2D$. Applying this inequality to (22) yields $p = O(|D|\log^6|D|\operatorname{llog}^8|D|)$, which then implies $v = O(\log^3|D|\operatorname{llog}^4|D|)$. □

We could obtain tighter bounds on p_{M} and v_{M} by modifying Algorithm 2.1 to only consider primes in $R_x \cap S_z$, but there is no reason to do so. Larger primes will be selected for S only when they improve the situation.

To prove the space bound in Theorem 1, we assume a time/space trade-off is made in the implementation of Algorithm 2.1. We control the space used to find the primes in S_z, by sieving within a suitably narrow window. This increases the running time by a negligible poly-logarithmic factor.

Lemma 4 (GRH)**.** *The expected running time of Algorithm* 2.1 *is $O(|D|^{1/2+\epsilon})$, using $O(|D|^{1/2}\log|D|\operatorname{llog}|D|)$ space.*

Proof. When computing S_z, it suffices to consider v up to an $O(\log^{3+\epsilon}|D|)$ bound, by Lemma 3 above. For each v we sieve the polynomial $f(t) = t^2 - v^2D$ to find $f(t) = 4p$ with p prime. The bound on p implies that we need only sieve to an $L = O(|D|^{1/2}\log^{3+\epsilon}|D|)$ bound on t. We may enumerate the primes up to L in $O(L \operatorname{llog} L)$ time using $O(\sqrt{L}\log L) = O(|D|^{1/4+\epsilon})$ space (we sieve with primes up to $\sqrt{L}$ to identify primes up to L using a window of size $\sqrt{L}$).

For each of the $\pi(L)$ primes $\ell \le L$, we compute a square root of $-v^2D$ modulo ℓ probabilistically, in expected time $O(\mathsf{M}(\log\ell)\log\ell)$, and use it to sieve $f(t)$. Here we sieve using a window of size $O(|D|^{1/2}\log|D| \operatorname{llog}|D|)$, recomputing each square root $O(\log^{2+\epsilon}|D|)$ times in order to achieve the space bound.

For each v, the total cost of computing square roots is $O(\pi(L)\log^{4+\epsilon}|D|)$, which dominates the cost of sieving. Applying $\pi(L) = O(L/\log L)$ and summing over v yields $O(|D|^{1/2}\log^{9+\epsilon}|D|)$, which dominates the time to select $S \subset S_z$.

To stay within the space bound, if we find that increasing z in step 3 by a factor of $1+\delta$ causes S_z to be too large (say, greater than $4b$ bits), we backtrack and instead increase z by a factor of $1+\delta/2$ and set $\delta \leftarrow \delta/2$. We increase z a total of $O(\log|D|)$ times (including all backtracking), and the lemma follows. □

In practice we don't actually need to make the time/space tradeoff described in the proof above. Heuristically we expect $p_{\mathrm{M}} = O(|D|\log^{1+\epsilon}|D|)$, and in this case all the primes in S_z can be found in a single pass with $L = O(|D|^{1/2}\log^{1/2+\epsilon}|D|)$.

We now show that all the precomputation steps in Algorithm 2 take negligible time and achieve the desired space bound. This includes selecting primes (Algorithm 2.1 in Section 3.3), computing polycyclic presentations (Algorithm 2.2 in Section 5.1), and CRT precomputation (Algorithm 2.3 in Section 6.2).

Lemma 5 (GRH)**.** *Steps* 1, 2, *and* 3 *of Algorithm* 2 *take* $O(|D|^{1/2+\epsilon})$ *expected time and use* $O(|D|^{1/2}(\log|D| + \log P)\operatorname{llog}|D|)$ *space.*

Proof. The complexity of step 1 is addressed by Lemma 4 above. By Proposition 6, step 2 performs $h(D)$ operations in $\mathrm{cl}(D)$, each taking $O(\log^2|D|)$ time [9]. Even if we compute a different presentation for every $v \le v_{\mathrm{M}}$, the total time is $O(|D|^{1/2+\epsilon})$. The table used by Algorithm 2.2 stores $h(D) = O(|D|^{1/2}\operatorname{llog}|D|)$ group elements, by bound (i), requiring $O(|D|^{1/2}\log|D|\operatorname{llog}|D|)$ space.

As described in Section 6.2, when $\log P \le \mu\log^3|D|$ the complexity of Algorithm 2.3 is $O(\mathsf{M}(\log M)\log^2 n)$ time and $O(\log M + n\log P)$ space, and we have

$$\log M = \sum_{p\in S}\log p \le n\log p_{\mathrm{M}} = O(|D|^{1/2}\log|D|\operatorname{llog}|D|),$$

according to bounds (iii) and (vi) above. As discussed in Section 6.3, the same time and space bounds for precomputation apply when $\log P > \mu\log^3|D|$. □

We next consider TestCurveOrder (Section 3.4), which is used by Algorithm 1.1 to find a curve in $\mathrm{Ell}_t(\mathbb{F}_p)$. We assume [64, Alg. 7.4] is used to implement the algorithm FastOrder which is called by TestCurveOrder.

Lemma 6. TestCurveOrder *runs in expected time* $O(\log^2 p \operatorname{llog}^2 p)$.

Proof. For $s = 0, 1$ the integer m_s computed by TestCurveOrder is the lcm of the orders of random points in $E_s(\mathbb{F}_p)$. By [64, Thm. 8.1] we expect that $O(1)$ points yield $m_s = \lambda(E_s(\mathbb{F}_p))$, the group exponent of $E_s(\mathbb{F}_p)$. For $p > 11$, Theorem 2 and Table 1 of [25] then imply $\mathcal{N} \subseteq \{N_0, N_1\}$, forcing termination. We thus expect to execute each step $O(1)$ times. We now bound the cost of steps 2-5:

2. The nonresidue used to compute $\tilde{E}$ can be probabilistically obtained using an expected $O(\log p)$ operations in $\mathbb{F}_p$, via Euler's criterion.
3. With E_s in the form $y^2 = f(x)$, we obtain a random point (x, y) by computing the square-root of $f(x)$ for random $x \in \mathbb{F}_p$, using an expected $O(\log p)$ operations in $\mathbb{F}_p$ to compute square roots (probabilistically).
4. Computing $Q = m_s P$ uses $O(\log p)$ group operations in $E_s(\mathbb{F}_p)$. The factorization of N_s/m_s is obtained by maintaining m_s in factored form. Implementing FastOrder via [64, Alg. 7.4] uses $O(\log p \operatorname{llog} p / \operatorname{lllog} p)$ group operations on $E_s(\mathbb{F}_p)$, by [64, Prop. 7.3].
5. The intersection of two arithmetic sequences can be computed with the extended Euclidean algorithm in time $O(\log^2 p)$, by [70, Thm. 3.13].

Step 4 dominates. The group operation in $E_s(\mathbb{F}_p)$ uses $O(1)$ operations in $\mathbb{F}_p$, each with bit complexity $O(\mathsf{M}(\log p))$, and this yields the bound of the lemma. □

We are now ready to bound the complexity of Algorithm 1 (Section 2), which computes $H_D \bmod p$ using Algorithm 1.1 (Section 3.4), Algorithm 1.2 (Section 4.1), and Algorithm 1.3 (Section 4.2).

Lemma 7 (GRH)**.** *For $p \in S$, Algorithm* 1 *computes $H_D \bmod p$ with an expected running time of $O(|D|^{1/2} \log^5 |D| \operatorname{llog}^3 |D|)$, using $O(|D|^{1/2} \log |D| \operatorname{llog} |D|)$ space.*

Proof. Ignoring the benefit of any torsion constraints, Algorithm 1.1 expects to sample $p/H(-v^2D) \leq z$ random curves over $\mathbb{F}_p$ to find $j \in \mathrm{Ell}_t(\mathbb{F}_p)$. The cost of testing a curve is $O(\log^2 p \operatorname{llog}^2 p)$, by Lemma 6, and this bound dominates the cost of any filters applied prior to calling TestCurveOrder.

Applying bound (v) on z and bound (vi) on p_{M} yields an overall bound of

$$O(|D|^{1/2} \log^5 |D| \operatorname{llog}^3 |D|) \tag{23}$$

on the expected running time of Algorithm 1.1, and it uses negligible space.

Algorithm 1.2 finds $j \in \mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$ in polynomial time if the conductor of D is small, and otherwise its complexity is bounded by the $O(p^{1/3}) = O(|D|^{1/3+\epsilon})$ complexity of Kohel's algorithm (under GRH). In either case it is negligible.

As shown in [4], the ERH yields an $O(\log^2 |D|)$ bound on the prime norms needed to generate $\mathrm{cl}(D)$, even if we exclude norms dividing v (at most $O(\operatorname{llog} |D|)$ primes). It follows that every optimal polycyclic presentation used by Algorithm 1.2 has norms bounded by $\ell_{\mathrm{M}} = O(\log^2 |D|)$. To bound the running time of Algorithm 1.3 we assume $\ell_i \nmid v$, since we use $\ell_i | v$ only when it improves performance.

The time to precompute each Φ_{ℓ_i} is $O(\ell_i^{3+\epsilon}) = O(\log^{6+\epsilon} |D|)$, by [28], and at most $O(\log |D|)$ are needed. These costs are negligible relative to the desired bound, as is the cost of reducing each Φ_{ℓ_i} modulo p. Applying the bound on ℓ_{M} and bound (vi) on p_{M}, each step taken by Algorithm 1.3 on an ℓ_i-isogeny cycle uses $O(\log^4 |D|)$ operations in $\mathbb{F}_p$, by (13). A total of h steps are required, and the bounds (i) on h and (vi) on p yield a bit complexity of $O(|D|^{1/2} \log^5 |D| \operatorname{llog}^{2+\epsilon} |D|)$ for Algorithm 1.3, using $O(h \lg p) = O(|D|^{1/2} \log |D| \operatorname{llog} |D|)$ space.

Step 4 of Algorithm 1 computes $\prod(X - j)$ over $j \in \mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$ via a product tree, using $O(\mathsf{M}(h) \log h)$ operations in $\mathbb{F}_p$ and space for two levels of the tree. Applying bound (i), this uses $O(|D| \log^{3+\epsilon} |D|)$ time and $O(|D|^{1/2} \log |D| \operatorname{llog} |D)$ space. □

The time bound in Lemma 7 may be improved to $O(|D|^{1/2}\log^5|D|\operatorname{llog}^2|D|)$ by arguing that a random point on a random elliptic curve over $\mathbb{F}_p$ has order greater than $4\sqrt{p}$ with probability $1-O(1/\log p)$.

Theorem 1 (GRH)**.** *Algorithm 2 computes $H_D \bmod P$ in $O(|D|\log^5|D|\operatorname{llog}^4|D|)$ expected time, using $O(|D|^{1/2}(\log|D|+\log P)\operatorname{llog}|D|)$ space.*

Proof. Lemma 5 bounds the cost of steps 1–3. As previously noted, if we have $P > M = \prod_{p\in S} p$, we set $P = M$ and compute H_D over $\mathbb{Z}$.

Algorithm 1 is called for each $p \in S$, of which there are $n = O(|D|^{1/2}\operatorname{llog}|D|)$, by bound (iii). Applying Lemma 7, Algorithm 1 computes $H_D \bmod p$ for all $p \in S$ within the time and space bounds stated in the theorem.

Recalling (18) from Section 6.2, for $\log P \le \mu\log^3|D|$ the total cost of updating the CRT sums via Algorithm 2.4 is bounded by

$$O\big(nh\mathsf{M}(\log P) + h\mathsf{M}(\log M + n\log n)\big). \tag{24}$$

We have $\log M \le n\log p_{\mathrm{M}} = O(|D|^{1/2}\log|D|\operatorname{llog}|D|)$, by bounds (iii) and (vi), thus (24) is bounded by $O(|D|\log^{3+\epsilon}|D|)$, using bound (i) on h. The cost of Algorithm 2.5 in step 5 is $O(h\mathsf{M}(\log P)) = O(|D|^{1/2+\epsilon})$, with $\log P = O(\log^3|D|)$. The space required is $O(h(\log|D|+\log P))$, which matches the bound in the theorem.

For $\log P > \mu\log^3|D|$, we apply the hybrid approach of Section 6.3, whose costs are bounded in (19). Using the bounds on $\log M$, n, and h, we again obtain an $O(|D|\log^{3+\epsilon}|D|)$ time for all CRT computations, and the space is as above. □

The CRT approach is particularly well suited to a distributed implementation; one simply partitions the primes in S among the available processors. The precomputation steps in Algorithm 2 have complexity $O(|D|^{1/2+\epsilon})$, under the GRH, and this is comparable to the complexity of Algorithm 1. Parallelism can be applied here, but in practice we are happy to repeat the precomputation on each processor.

When $\log P$ is polynomially bounded in $\log|D|$, the postcomputation can be performed in time $O(|D|^{1/2+\epsilon})$ by aggregating the CRT sums, with the final result $H_D \bmod P$ available on a single node. When P is larger, as when computing H_D over $\mathbb{Z}$, we may instead have each processor handle the postcomputation for a subset of the coefficients of H_D, leaving the final result distributed among the processors.

We do not attempt a detailed analysis of the parallel complexity here, but note the following corollary, which follows from the discussion above.

Corollary 1 (GRH)**.** *There is a parallel algorithm to compute $H_D \bmod P$ on $O(|D|^{1/2+\epsilon})$ processors that uses $O(|D|^{1/2+\epsilon})$ time and space per processor.*

7.1. A heuristic analysis.

To obtain complexity estimates that better predict the actual performance of Algorithms 1 and 2, we consider a naïve probabilistic model. We assume that each positive integer m is prime with probability $1/\log m$, and that for each prime $\ell \nmid D$ we have $(\frac{D}{\ell}) = 1$ with probability 1/2. For a prime ℓ with $(\frac{D}{\ell}) = 1$ we further assume that if $\alpha, \alpha^{-1} \in \mathrm{cl}(D)$ are distinct classes containing an ideal of norm ℓ, then α corresponds to a random element of $\mathrm{cl}(D)$ uniformly distributed among the elements of order greater than 2. Most critically, we suppose that all these probabilities are independent. This last assumption is obviously false, but when applied on a large scale this model yields empirically accurate predictions.

Compared to the GRH-based analysis, these assumptions do not change the space complexity, nor bounds (i)–(iii), but significantly improve bounds (iv)–(vii).

(**H**) Our heuristic model predicts the following:

(iv) $\boldsymbol{\ell}_{\mathrm{M}} = O(\log^{1+\epsilon}|D|)$.
(v) $\boldsymbol{z} = O(|D|^{1/2}\log^{1/2+\epsilon}|D|)$.
(vi) $\boldsymbol{p}_{\mathrm{M}} = O(|D|\log^{1+\epsilon}|D|)$.
(vii) $\boldsymbol{v}_{\mathrm{M}} = O(\log^{1/2+\epsilon}|D|)$.

Applying these to the analysis of Section 7 yields an $O(|D|\log^{3+\epsilon}|D|)$ bound on the expected running time of Algorithm 2, matching the heuristic result in [5].

It is claimed in [5, §5.4] that applying the bounds (i) and (ii) to [27, Thm. 1.1] also yields a heuristic complexity of $O(|D|\log^{3+\epsilon}|D|)$ when using the floating-point method to compute H_D. This is incorrect; the implied bound is actually $O(|D|\log^{4+\epsilon}|D|)$ (as confirmed by the author of [27]).

One may reasonably question how accurate our $O(|D|\log^{3+\epsilon}|D|)$ estimate is in practice, since it assumes that the Fast Fourier Transform (FFT) is used for all multiplications. The cost $\mathsf{M}(n)$ arises in three distinct contexts:

(a) The cost of operations in $\mathbb{F}_p$ is bounded by $O(\mathsf{M}(\log p))$.
(b) Finding a root of $\Phi_\ell(X, j_i)/(X - j_{i-1})$ uses $O(\mathsf{M}(\ell)\log p)$ $\mathbb{F}_p$-operations.
(c) Computing $\prod(X - j)$ uses $O(\mathsf{M}(h)\log h)$ $\mathbb{F}_p$-operations.

In case (a) we actually expect $\lg p_{\mathrm{M}}$ to be smaller than the word size of our CPU, so multiplications in $\mathbb{F}_p$ effectively have unit cost. For (b), ℓ is typically in the range where either schoolbook or Karatsuba-based multiplication should be used. It is only in case (c) that FFT-based algorithms may be profitably applied.

In order to better estimate the running time of Algorithm 1 (which effectively determines the running time of Algorithm 2) we break out the cost of each step, expressing all bounds in terms of $\mathbb{F}_p$-operations.

Table 1. (**H**) Heuristic complexity of Algorithm 1

step	Complexity ($\mathbb{F}_p$-operations)
1. Find $j \in \mathrm{Ell}_t(\mathbb{F}_p)$	$O(|D|^{1/2}\log^{3/2+\epsilon}|D|)$
2. Find $j' \in \mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$	negligible
3. Enumerate $\mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$	$O(|D|^{1/2}\log^{1+\omega+\epsilon}|D|)$
4. Compute $\prod_{j\in\mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)}(X - j)$	$O(|D|^{1/2}\log^{2+\epsilon}|D|)$

The value of ω depends on our estimate for $\mathsf{M}(\ell)$. One can find values of D in the feasible range where ℓ_{M} is over 300 (see [40, 41]), and here it is reasonable to assume $\mathsf{M}(\ell) = \ell^\omega$ with $\omega = \lg 3 \approx 1.585$. In the worst case, step 3 dominates.

However, the critical parameter is ℓ_1, the least cost ℓ_i used by Algorithm 1.3. If $\ell_1 \nmid D$ we expect it to be used in the overwhelming majority of the steps taken by Algorithm 1.3. As with ℓ_{M}, it is possible to find feasible D for which ℓ_1 is fairly large (over 100), but such cases are extremely rare. If we average over D in some large interval, our heuristic model predicts $\ell_1 = O(1)$ (in fact $\mathbf{E}[\ell_1] < 4$). We typically have $\mathsf{M}(\ell_1) = O(1)$ and use $\omega = 0$. In almost all cases, step 4 dominates.

The relative cost of step 4 is not significant for small $|D|$, due to the excellent constant factors in the algorithms available for polynomial multiplication, but its asymptotic behavior becomes evident as $|D|$ grows (see Tables 3 and 4).

8. Computational results

To assess the performance of the new algorithm in a practical application, we used it to construct pairing-friendly curves suitable for cryptographic use, a task that often requires large discriminants. We constructed ordinary elliptic curves of prime order and embedding degree k over a prime field $\mathbb{F}_q$ such that either

$$k = 6 \text{ and } 170 < \lg q < 192 \qquad \text{or} \qquad k = 10 \text{ and } 220 < \lg q < 256.$$

These parameters were chosen using the guidelines in [31] and have particularly desirable performance and security characteristics. For additional background on pairing-based cryptography we refer to [20, Ch. 24].

To obtain suitable discriminants we used algorithms in [44] (for $k = 6$) and [30] (for $k = 10$) that were optimized to search for q within a specified range. This produced a set $\mathcal{D}_{\text{PF}}$ of nearly 2000 fundamental discriminants (1722 with $k = 6$ and 254 with $k = 10$), with $|D|$ ranging from about 10^7 to just over 10^{13} (almost all greater than 10^{10}). We selected 200 representative discriminants from $\mathcal{D}_{\text{PF}}$ for our tests, including those that potentially posed the greatest difficulty, due to an unusually large value of ℓ_1 or $h(D)$.

To each selected discriminant we applied the CM method, using Algorithm 2 to compute $H_D \bmod P$ (with $P = q$). After finding a root j of $H_D(X)$ over $\mathbb{F}_q$, we construct an elliptic curve E with this j-invariant and ensure that the trace of E has the correct sign.[1]

8.1. **Implementation.** The algorithms described in this paper were implemented using the GNU C/C++ compiler [63] and the GMP library [33] on a 64-bit Linux platform. Multiplication of large polynomials was handled by the zn_poly library developed by Harvey [36], based on the algorithm in [35].

The hardware platform included sixteen 2.8 GHz AMD Athlon processors, each with two cores. Up to 32 cores were used in each test (with essentially linear speedup), but for consistency we report total cpu times, not elapsed times. Memory utilization figures are per core and can be achieved using a single core.

8.2. **Distribution of test discriminants.** To construct a curve of odd order over a field of odd characteristic we must have $D \equiv 5 \bmod 8$, and this necessarily applies to $D \in \mathcal{D}_{\text{PF}}$. We then have $(\frac{D}{2}) = -1$, which implies $\ell_1 \geq 3$ and also tends to make $h(D)$ smaller than it would be for an arbitrary discriminant. Averaging over all discriminants up to an asymptotically large bound, we expect

$$L(1, \chi_D) = \frac{\pi h(D)}{\sqrt{|D|}} \quad \longrightarrow \quad C\pi^2/6 \approx 1.45,$$

where $C = \prod_p \bigl(1 - 1/(p^2(p+1))\bigr)$; see [18, p. 296] (and see [40] for actual data). Among the 1722 discriminants we found for $k = 6$, the average value of $L(1, \chi_D)$ is about 0.55, close to the typical value for $D \equiv 5 \bmod 8$. For $k = 10$ we have the further constraint $\ell_1 \geq 7$, and the average value of $L(1, \chi_D)$ is approximately 0.40.

While we regard the discriminants in $\mathcal{D}_{\text{PF}}$ as representative for the application considered, in order to assess the performance of Algorithm 2 in more extreme cases we also conducted tests using discriminants with very large values of $L(1, \chi_D)$. These results are presented in Section 8.5.

[1] One may apply the method in [56], or simply compute NQ for a nonzero point $Q \in E(\mathbb{F}_q)$, where N is the desired (prime) order of $E(\mathbb{F}_q)$, and switch to a quadratic twist of E if $NQ \neq 0$.

TABLE 2. Example computations

	Example 1	Example 2	Example 3
$\|D\|$	$13,569,850,003$	$11,039,933,587$	$12,901,800,539$
$h(D)$	20,203	11,280	54,706
$L(1,\chi_D)$	0.54	0.34	1.51
b	2,272,566	1,359,136	5,469,778
n	63,682	39,640	142,874
z	755,637	734,040	905,892
$\lceil \lg p_{\mathrm{M}} \rceil$	40	38	43
v_{M}	12	8	32
$(\ell_1^{r_1},\ldots,\ell_k^{r_k})$	(7^{20203})	$(17^{1128},19^{10})$	$(3^{27038},5^2)$
step 1	0.0s	0.0s	0.0s
step 2	1.2s	0.5s	4.0s
step 3	0.6s	0.3s	2.0s
step 4	23,300s	26,000s	61,000s
step 5	0.0s	0.0s	0.0s
$(T_{\mathrm{f}},T_{\mathrm{e}},T_{\mathrm{b}})$	(57,32,11)	(51,47,2)	(53,20,27)
throughput	2.0Mb/s	0.6Mb/s	4.9Mb/s
memory	3.9MB	2.1MB	9.4MB
total data	5.7GB	1.9GB	37GB
Solve $H_D(X)=0$ over $\mathbb{F}_q$	127s	86s	332s

(2.8 GHz AMD Athlon)

8.3. **Examples.** Table 2 summarizes computations for three discriminants of comparable size, with $|D| \approx 10^{10}$. These represent a typical case (Example 1) and two "worst" cases (Examples 2 and 3). The parameters appearing in the top section of the table are as defined in Section 7. The next section of the table contains timings for each step of Algorithm 2.

As predicted by the asymptotic analysis, essentially all of the time is spent in step 4, which calls Algorithm 1 for each prime $p \in S$. There are three principal components in the running time of Algorithm 1:

T_{f}: time spent in step 1 finding a curve in $\mathrm{Ell}_t(\mathbb{F}_p)$;
T_{e}: time spent in step 3 enumerating $\mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)$;
T_{b}: time spent in step 4 building $H_D(X) = \prod_{j\in\mathrm{Ell}_{\mathcal{O}}(\mathbb{F}_p)}(X-j) \bmod p$.

These are listed in Table 2 as percentages of the total time T. The time spent elsewhere $(T - T_{\mathrm{f}} - T_{\mathrm{e}} - T_{\mathrm{b}})$ is well under 1% of T.

The third section in Table 2 lists the throughput, memory utilization, and total data processed during the computation.[2] The total data is defined as the product of the number of coefficients $h(D)$ and the height bound b. This approximates the total size of H_D, typically overestimating it by about 10% (the actual sizes of H_D for the three examples are 5.3GB, 1.8GB and 34GB, respectively). The throughput is then the total data divided by the total time. Memory figures include all working storage and overhead due to data alignment (to word boundaries and to powers of 2 in FFT computations), but exclude the fixed operating system overhead of about

[2] The suffixes Mb, MB, and GB indicate 10^6 bits, 10^6 bytes, and 10^9 bytes, respectively.

TABLE 3. Performance for typical $D \in \mathcal{D}_{\text{PF}}$

$\lvert D\rvert$	$h(D)$	cpu secs	$(T_{\text{f}}, T_{\text{e}}, T_{\text{b}})$	Mb/s	memory	data
$116,799,691$	$2,112$	156	(65,28,7)	2.6	0.5MB	52MB
$1,218,951,379$	$6,320$	1,650	(64,26,8)	2.5	1.1MB	520MB
$13,596,850,003$	$20,203$	23,400	(57,33,10)	2.0	3.9MB	5.7GB
$126,930,891,691$	$56,282$	195,000	(66,22,12)	2.0	9.5MB	50GB
$1,009,088,517,019$	$181,584$	2,160,000	(64,20,16)	2.0	34MB	535GB
$10,028,144,961,139$	$521,304$	20,600,000	(63,20,17)	1.9	84MB	5.0TB

(2.8 GHz AMD Athlon)

4MB. The last row of the table lists the time to find a root of H_D over $\mathbb{F}_q$, although this task is not actually performed by Algorithm 2.

Example 1 represents a typical case: $L(1, \chi_D)$ is close to the mean of 0.55, and $\ell_1 = 7$ is just above the median of 5 (over $D \in \mathcal{D}_{\text{PF}}$). Example 2 has an unusually large $\ell_1 = 17$ (exceeded by fewer than 1% of $D \in \mathcal{D}_{\text{PF}}$), while Example 3 has an unusually large $L(1, \chi_D) \approx 1.51$ (exceeded by fewer than 1% of $D \in \mathcal{D}_{\text{PF}}$).

In Example 2, the large ℓ_1 increases T_{e} substantially, despite the smaller $h(D)$. The smaller $L(1, \chi_D)$ tends to increase the running time of individual calls to Algorithm 1.1, but at the same time n decreases so that overall T_{f} decreases slightly. The smaller values of $h(D)$ and n both serve to decrease T_{b} significantly.

In Example 3 the large $L(1, \chi_D)$ decreases the cost of individual calls to Algorithm 1.1, but increases n substantially so that overall T_{f} increases. However, T_{e} and T_{b} increase even more, especially T_{b}. Despite the longer running time, this scenario results in the highest throughput of the three examples.

8.4. **Scaling.** Table 3 summarizes the performance of Algorithm 2 for $D \in \mathcal{D}_{\text{PF}}$ ranging over six orders of magnitude. We selected examples whose performance was near the median value for discriminants of comparable size. We note the quasi-linear growth of T and the increasing value of T_{b} as a percentage of T, consistent with our heuristic prediction that this component is asymptotically dominant.

Up to 32 cores were applied to the computations in Table 3. In all but the smallest example we can effectively achieve a 32x speedup. The actual elapsed time for the largest discriminant was about 8 days, while the second largest took less than a day. As suggested by Corollary 1, these computations could be usefully distributed across many more processors. The low memory requirements provide headroom for much larger computations: each of our cores had 2GB of memory, but less than 100MB was used.

Below is an example of a curve constructed using $D = -10,028,144,961,139$, the largest discriminant listed in Table 3. The elliptic curve

$$y^2 = x^3 - 3x + 3338561401570133202017008597803337396411439360229378547$$

has embedding degree 6 over the finite field $\mathbb{F}_q$ with

$$q = 30518311673028635209000068713843412774183984182022701057.$$

This curve has prime order $N = q + 1 - t$, where

$$t = 5524338120809463560527395583.$$

Table 4. Performance when $L(1,\chi_D)$ is large.

$\lvert D\rvert$	$h(D)$	cpu secs	$(T_{\mathrm{f}}, T_{\mathrm{e}}, T_{\mathrm{b}})$	Mb/s	memory	data
$2,093,236,031$	100,000	98,800	(25,19,56)	7.5	18MB	93GB
$8,364,609,959$	200,000	472,000	(24,17,59)	6.8	36MB	400GB
$17,131,564,271$	300,000	1,240,000	(20,15,65)	5.9	61MB	920GB
$30,541,342,079$	400,000	2,090,000	(21,16,63)	6.4	71MB	1.7TB
$42,905,564,831$	500,000	3,050,000	(22,17,61)	6.9	81MB	2.6TB
$67,034,296,559$	600,000	5,630,000	(18,14,68)	5.6	121MB	3.9TB
$82,961,887,511$	700,000	7,180,000	(19,14,67)	5.9	132MB	5.3TB
$113,625,590,399$	800,000	9,520,000	(19,15,66)	5.9	142MB	7.1TB
$133,465,791,359$	900,000	11,500,000	(20,15,65)	6.2	152MB	9.0TB
$170,868,609,071$	1,000,000	14,200,000	(20,16,64)	6.3	163MB	11.2TB

(2.8 GHz AMD Athlon)

There are a total of $h(D) = 521,304$ nonisomorphic curves with the same order that may be constructed using $H_D \bmod q$. A complete list of curves for all the discriminants tested is available at `http://math.mit.edu/~drew`.

8.5. **Discriminants with large** $L(1,\chi_D)$**.** Table 4 shows the performance of Algorithm 2 on discriminants specifically chosen to make $L(1,\chi_D)$ extremely large, between 6.8 and 7.8. These discriminants are not in $\mathcal{D}_{\mathrm{PF}}$ and are likely the smallest possible for the class numbers listed (but we do not guarantee this). In each case we computed H_D modulo a 256-bit prime P. The timings would not change significantly for larger P, but the space would increase.

The first discriminant $D = -2,093,236,031$ in Table 4 also appears in Table 1 of [27]. Scaled to the same processor speed, Algorithm 2 computes $H_D \bmod P$ using less than half the cpu time spent by the floating-point approximation method to compute a class polynomial over $\mathbb{Z}$ for the same D (this polynomial would then need to be reduced mod P in order to apply the CM method). Most significantly, the memory required is about 20MB versus 5GB.

This comparison is remarkable, given that the height bound $b = 7,338,789$ for H_D is nearly 28 times larger than the 264,727 bits of precision used in [27], where the class polynomial for the double-eta quotient $\mathfrak{w}_{3,13}$ was computed instead of the Hilbert class polynomial. The difference in throughput is thus much greater than the difference in running times: 7.5Mb/s versus 0.10Mb/s.

Appendix 1

This appendix proves Lemma 8, which bounds the coefficients of the Hilbert class polynomial $H_D(X)$, and Lemma 9, which bounds the Hurwitz class number $H(-v^2D)$ in terms of v and $H(-D)$.

Let B denote an upper bound on the absolute values of the coefficients of $H_D(X)$. In the literature one finds many values for B (or $\log B$), but due to an unfortunate series of typographical errors, most are either incorrect [5, p. 285], exponentially larger than necessary ([2, Eq. 22] and [11, p. 151]), or heuristics that do hold for all D ([1, Eq. 3.1] and [12, p. 2431]). In [27, Thm. 1.2], Enge gives a rigorous and fully explicit value for B that is empirically accurate to within a constant factor, but still larger than desirable for practical application. Provided one is prepared to enumerate the elements of $\mathrm{cl}(D)$, a much tighter bound is given by the lemma below, whose proof is derived directly from Enge's analysis in [27, §4].

Lemma 8. *For a quadratic discriminant $D < 0$, let $(a_1, b_1, c_1), \ldots, (a_h, b_h, c_h)$ be the sequence of reduced, primitive, binary quadratic forms of discriminant D with $0 < a_1 \le \cdots \le a_h$, where $h = h(D)$. Let $M_k = \exp(\pi\sqrt{|D|}/a_k) + C$, where $C = 2114.567$. Then the coefficients of $H_D(X)$ have absolute values bounded by*

$$B = \binom{h}{m} M_h^{-m} \prod_{k=1}^{h} M_k,$$

where $m = \left\lfloor \frac{h+1}{M_h+1} \right\rfloor$. We also have $\log B = O(|D|^{1/2} \log^2 |D|)$, and under the GRH, $\log B = O(|D|^{1/2} \log |D| \operatorname{llog} |D|)$.

Proof. We may write H_D as

$$H_D(X) = \prod_{k=0}^{h} \bigl(X - j(\tau_k)\bigr),$$

where $\tau_k = (-b_k + \sqrt{D})/2a_k$. With $q_k = e^{2\pi i \tau_k}$, we have $M_k = |1/q_k| + C$, where the constant C bounds $|j(\tau_k) - 1/q_k|$, as shown in [27, p. 1094]. Thus $|j(\tau_k)| \le M_k$, and the absolute value of the coefficient of X^n in $H_D(X)$ is bounded by

$$B_n = \binom{h}{n} \prod_{k=1}^{h-n} M_k. \tag{25}$$

We now argue that $B_n \le B$. For $n > m$ we have $n > (h+1)/(M_h+1)$ and

$$\binom{h}{n} \bigg/ \binom{h}{n-1} = \frac{h-n+1}{n} < M_h.$$

This implies $B_n < B_{n-1}$. For $0 < n \le m$ we have $\binom{h}{n}/\binom{h}{n-1} \ge M_h$, which implies

$$B_0 \le B_1 M_h / M_h \le B_2 M_{h-1} M_h / M_h^2 \le \cdots \le B_m M_{h-m+1} \cdots M_h / M_h^m = B.$$

It follows that B bounds every B_n.

The bound $\log B = O(|D|^{1/2} \log^2 |D|)$ follows from $h = O(|D|^{1/2} \log |D|)$, as proven in [59], and the bound $\sum_k \frac{1}{a_k} = O(\log^2 |D|)$, as proven in [58, Lemma 2.2]. As shown in [5, Lemma 2], under the GRH the bound $\sum_k \frac{1}{a_k} = O(\log |D| \operatorname{llog} |D|)$ follows from [52], which yields $\log B = O(|D|^{1/2} \log |D| \operatorname{llog} |D|)$. □

In practice the bound given by Lemma 8 is close to, and often better than, the heuristic bound $B = \binom{h}{\lfloor h/2 \rfloor} \exp(\pi\sqrt{|D|} \sum_k \frac{1}{a_k})$ that is sometimes used, even though the latter bound is not actually valid for all D (such as $D = -99$).

Lemma 9. *Let D be a negative discriminant, let $v \ge 2$ be an integer, and let x be the largest prime for which $\prod_{p \le x} p \le v$, where p ranges over primes. Let $H(n)$ denote the Hurwitz class number. The following inequalities hold:*

$$1 \le \frac{H(-v^2 D)}{v H(-D)} \le \prod_{p \le x} \frac{p+1}{p-1} < 11 \operatorname{llog}^2(v+4).$$

Proof. Let u be the conductor of D, so that $D = u^2D_0$. Then

$$H(-v^2D) = H(-(uv)^2D_0) = \sum_{d|uv} \frac{2h(d^2D_0)}{w(d^2D_0)}, \tag{26}$$

where $w(d^2D_0) = |\mathcal{O}^*_{d^2D_0}|$ is 2, 4, or 6 [18, Lemma 5.3.7]. We also have [18, p. 233]

$$\frac{h(d^2D_0)}{w(d^2D_0)} = \frac{h(D_0)}{w(D_0)} d \prod_{p|d} \left(1 - \frac{\chi_p}{p}\right),$$

where $\chi_p = \left(\frac{D_0}{p}\right)$ is -1, 0, or 1. Regarding D_0 as fixed, we note that

$$H(-n^2D_0) = \frac{2h(D_0)}{w(D_0)} \sum_{d|n} d \prod_{p|d} \left(1 - \frac{\chi_p}{p}\right)$$

is a multiplicative function of n, which yields

$$H(-n^2D_0) = \frac{2h(D_0)}{w(D_0)} \prod_p \left(1 + \left(p^{\nu_p(n)} - 1\right)(p - \chi_p)/(p-1)\right),$$

where $\nu_p(n)$ is the p-adic valuation. From (26) we obtain

$$\frac{H(-v^2D)}{vH(-D)} = \frac{\prod_p \left(1 + \left(p^{\nu_p(u)+\nu_p(v)} - 1\right)(p - \chi_p)/(p-1)\right)}{v\prod_p \left(1 + \left(p^{\nu_p(u)} - 1\right)(p - \chi_p)/(p-1)\right)}. \tag{27}$$

Fixing $D = u^2D_0$, we regard (27) as a multiplicative function of v. For $v = p^k$:

$$\frac{H(-p^2kD)}{p^kH(-D)} = \frac{\left(1 + \left(p^{\nu_p(u)+k} - 1\right)(p - \chi_p)/(p-1)\right)}{p^k\left(1 + \left(p^{\nu_p(u)} - 1\right)(p - \chi_p)/(p-1)\right)}.$$

This value is minimized when $\chi_p = 1$, in which case it is 1, yielding the first inequality in the lemma. It is maximized when $\chi_p = -1$, in which case one finds

$$\frac{\left(1 + \left(p^{\nu_p(u)+k} - 1\right)(p+1)/(p-1)\right)}{p^k\left(1 + \left(p^{\nu_p(u)} - 1\right)(p+1)/(p-1)\right)} \le \frac{p+1}{p-1},$$

for all nonnegative integers k and $\nu_p(u)$. We thus obtain from (27)

$$\frac{H(-v^2D)}{vH(-D)} \le \prod_{p|v} \frac{p+1}{p-1} \le \prod_{p \le x} \frac{p+1}{p-1},$$

proving the second inequality in the lemma. To prove the third inequality, we first note that for $v \ge \prod_{p\le x} p$ the inequality holds for each prime $x < 41$, by a machine calculation, so we assume $x \ge 41$. We then have

$$\log \prod_{p\le x} \frac{p+1}{p-1} = \sum_{p\le x} \log\left(1 + \frac{2}{p-1}\right) \le \sum_{p\le x} \frac{2}{p-1} = 2\sum_{p\le x} \frac{1}{p} + 2\sum_{p\le x} \frac{1}{p(p-1)}.$$

We now apply the bound $\sum_{p\le x}\frac{1}{p} < \operatorname{llog} x + B_1 + 1/(\log x)^2$ from [55, 3.20], where $B_1 = 0.261497\ldots$, and also $\sum_p \frac{1}{p(p-1)} = 0.773156\ldots$ from [19] to obtain

$$\log \prod_{p\le x}\frac{p+1}{p-1} < 2\operatorname{llog} x + 2.218,$$

valid for $x \ge 41$. This yields $\prod_{p\le x}\frac{p+1}{p-1} < 9.189\cdot\log^2 x$. We also have the bound $x(1-1/\log x) < \sum_{p\le x}\log p$, valid for $x\ge 41$, by [55, 3.16], which implies

$$\prod_{p\le x}\frac{p+1}{p-1} < 9.189\cdot\log^2(1.369\cdot\log v).$$

For $x \ge 41$ we have $\log v > 30$, and the RHS is then smaller than $11\operatorname{llog}^2(v+4)$. □

Appendix 2

Here we list some of the torsion constraints used to accelerate the search for an elliptic curve $E/\mathbb{F}_p$ with $p+1\pm t$ points, as described in Section 3. Each constraint has the form $m = a\cdot b\cdot N$, where a is a power of 2 and b is a power of 3. Curves with a point of order N are generated using a plane model for $X_1(N)$ as in [65] and are then filtered to ensure that the constraints implied by a and b are also met. When a or b is expressed in exponential notation, it is meant to control the exact power of 2 or 3 that divides $\#E$. The torsion constraint $14 = 2^0\cdot 3^0\cdot 14$, for example, indicates that $\#E$ is divisible by 14 but not divisible by 3 or 4.

Efficient methods for analyzing the Sylow 2-subgroup of $E(\mathbb{F}_p)$ are considered in [53, 65], and for 3-torsion we use the 3-division polynomial [71, § 3.2]. For the sake of brevity, here we consider constraints on the Sylow 2-subgroup only up to 4-torsion, but one may obtain minor improvements using 2^k-torsion for larger k.

The benefit of each constraint is computed as $1/r$, where r is the proportion of elliptic curves $E/\mathbb{F}_p$ that satisfy the constraint. We derive r using [38, Thm. 1.1], under the simplifying assumption that if N divides $\#E$, then $E(\mathbb{F}_p)$ contains a point of order N (necessarily true when the square part of N is coprime to $p-1$). A more precise estimate may be obtained from [32, Thm. 3.15]. Table 5 assumes that $p\equiv 1 \bmod 3$ and $p\not\equiv 1 \bmod \ell$ for primes $\ell > 3$ dividing N. It is easily adjusted to other cases via [38, Thm. 1.1]; this will change the rankings only slightly.

The cost of each constraint was determined empirically (and is somewhat implementation dependent). For a random set of primes p of suitable size (30-50 bits) we measured the average time to: (1) generate a curve $E/\mathbb{F}_p$ satisfying the constraint, (2) obtain a random point $P\in E(\mathbb{F}_p)$, and (3) compute the points $(p+1)P$ and tP. This is compared to the cost of (2) and (3) alone (the "null case" for Algorithm 1.1, excluding TestCurveOrder which is rarely called). The parametrizations of [3] combine (1) and (2), enabling a cost of less than 1.0 in some cases.

The rankings in Table 5 assume each constraint is applicable to both $N_0 = p+1-t$ and $N_1 = p+1+t$; if not, the effective ratio is about half the listed value (9/16, on average). For given values of p and t, we thus consider three possible constraints, one satisfied by N_0, one by N_1, and one by both, and then pick the best of the three.

Table 5. Ranking of m-torsion constraints (for $p \equiv 1 \bmod 3$).

m	torsion	benefit	cost	ratio	m	torsion	benefit	cost	ratio
33	$2^0 \cdot 3 \cdot 11$	80.0	2.3	34.3	44	$4 \cdot 11$	24.0	1.8	13.0
39	$2^0 \cdot 3 \cdot 13$	96.0	3.0	31.5	93	$2^0 \cdot 3 \cdot 31$	240.0	18.9	12.7
51	$2^0 \cdot 3 \cdot 17$	128.0	4.4	29.0	34	$2^1 \cdot 17$	64.0	5.0	12.7
15	$2^0 \cdot 15$	32.0	1.2	26.4	28	$2 \cdot 14$	14.4	1.2	12.4
11	$2^0 \cdot 11$	30.0	1.2	25.9	52	$4 \cdot 13$	28.8	2.4	12.2
57	$2^0 \cdot 3 \cdot 19$	144.0	5.7	25.4	18	$2^0 \cdot 18$	26.2	2.2	12.0
66	$2^1 \cdot 3 \cdot 11$	106.7	4.3	24.7	36	$2 \cdot 18$	15.7	1.3	12.0
21	$2^0 \cdot 3 \cdot 7$	48.0	2.1	23.1	68	$4 \cdot 17$	38.4	3.4	11.2
69	$2^0 \cdot 3 \cdot 23$	176.0	7.8	22.4	38	$2^1 \cdot 19$	72.0	6.7	10.8
78	$2^1 \cdot 3 \cdot 13$	128.0	5.8	22.0	10	$2^0 \cdot 10$	16.0	1.5	10.7
13	$2^0 \cdot 13$	36.0	1.6	21.8	174	$2^1 \cdot 3 \cdot 29$	298.7	28.6	10.4
9	$2^0 \cdot 9$	19.6	1.0	20.2	20	$2 \cdot 10$	9.6	0.9	10.4
102	$2^1 \cdot 3 \cdot 17$	170.7	8.5	20.0	348	$4 \cdot 3 \cdot 29$	179.2	18.0	9.9
42	$2^0 \cdot 3 \cdot 14$	64.0	3.2	20.0	76	$4 \cdot 19$	43.2	4.4	9.9
7	$2^0 \cdot 7$	18.0	0.9	19.6	46	$2^1 \cdot 23$	88.0	9.2	9.5
132	$4 \cdot 3 \cdot 11$	64.0	3.3	19.2	29	$2^0 \cdot 29$	84.0	8.9	9.5
17	$2^0 \cdot 17$	48.0	2.5	19.0	48	$3 \cdot 16$	21.3	2.3	9.4
156	$4 \cdot 3 \cdot 13$	76.8	4.2	18.2	3	$2^0 \cdot 3$	8.0	0.9	9.2
204	$4 \cdot 3 \cdot 17$	102.4	5.9	17.5	92	$4 \cdot 23$	52.8	6.0	8.8
114	$2^1 \cdot 3 \cdot 19$	192.0	11.0	17.4	12	12	6.4	0.7	8.8
30	$2^1 \cdot 15$	42.7	2.5	16.9	186	$2^1 \cdot 3 \cdot 31$	320.0	36.9	8.7
84	$2 \cdot 3 \cdot 14$	38.4	2.3	16.5	31	$2^0 \cdot 31$	90.0	11.5	7.8
19	$2^0 \cdot 19$	54.0	3.3	16.4	6	$2^0 \cdot 6$	10.7	1.4	7.4
22	$2^1 \cdot 11$	40.0	2.5	16.2	16	16	8.0	1.1	7.2
228	$4 \cdot 3 \cdot 19$	115.2	7.4	15.7	58	$2^1 \cdot 29$	112.0	17.4	6.4
87	$2^0 \cdot 3 \cdot 29$	224.0	14.5	15.5	116	$4 \cdot 29$	67.2	11.0	6.1
138	$2^1 \cdot 3 \cdot 23$	234.7	15.3	15.4	8	8	4.0	0.7	5.9
26	$2^1 \cdot 13$	48.0	3.3	14.4	62	$2^1 \cdot 31$	120.0	23.0	5.2
23	$2^0 \cdot 23$	66.0	4.7	14.2	124	$4 \cdot 31$	72.0	14.2	5.1
276	$4 \cdot 3 \cdot 23$	140.8	10.0	14.0	2	$2^0 \cdot 2$	4.0	0.9	4.3
14	$2^0 \cdot 14$	24.0	1.7	13.8	4	4	2.4	0.6	3.8
60	$4 \cdot 15$	25.6	1.9	13.5	1	$2^0 \cdot 1$	3.0	0.8	3.7
5	$2^0 \cdot 5$	12.0	0.9	13.0					

Dominated constraints are not listed, e.g. $3 \cdot 4 \cdot 31$ is always inferior to 12.

Acknowledgments

The author thanks Daniel J. Bernstein for initially suggesting this project, and also Gaetan Bisson, Reinier Bröker, Andreas Enge, David Harvey, Tanja Lange, and Kristin Lauter for their support and feedback on an early draft of this paper. The author is also grateful to the referee for a careful reading and many helpful suggestions.

References

1. Amod Agashe, Kristin Lauter, and Ramaranthnam Venkatesan, *Constructing elliptic curves with a known number of points over a prime field*, High Primes and Misdemeanours: Lectures in Honour of the 60th Birthday of Hugh Cowie Williams (A. J. van der Poorten and A. Stein, eds.), Fields Institute Communications, vol. 41, AMS, 2004, pp. 1–17. MR2075643 (2005m:11112)
2. A.O.L. Atkin and François Morain, *Elliptic curves and primality proving*, Mathematics of Computation **61** (1993), 29–68. MR1199989 (93m:11136)
3. ______, *Finding suitable curves for the elliptic curve method of factorization*, Mathematics of Computation **60** (1993), 399–405. MR1140645 (93k:11115)
4. Eric Bach, *Explicit bounds for primality testing and related problems*, Mathematics of Computation **55** (1990), no. 191, 355–380. MR1023756 (91m:11096)
5. 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. MR2467854 (2009j:11200)
6. Daniel J. Bernstein, *Detecting perfect powers in essentially linear time, and other studies in computational number theory*, Ph.D. thesis, University of California at Berkeley, 1995.
7. ______, *Multidigit modular multiplication with the explicit Chinese remainder theorem*, 1995, Chapter 4 of the author's Ph.D. thesis, available at `http://cr.yp.to/papers.html#mmecrt`.
8. ______, *Modular exponentiation via the explicit Chinese Remainder Theorem*, Mathematics of Computation **76** (2007), 443–454. MR2261030 (2007f:11142)
9. Ingrid Biehl and Johannes Buchmann, *An analysis of the reduction algorithms for binary quadratic forms*, Voronoi's Impact on Modern Science (P. Engel and H. Syta, eds.), Institute of Mathematics, Kyiv, 1998, available at `http://www.cdc.informatik.tu-darmstadt.de/reports/TR/TI-97-26.ps.gz`, pp. 71–98.
10. Gaetan Bisson and Andrew V. Sutherland, *Computing the endomorphism ring of an ordinary elliptic curve over a finite field*, Journal of Number Theory (2009), to appear, `http://arxiv.org/abs/0902.4670`.
11. Ian Blake, Gadiel Seroussi, and Nigel Smart, *Elliptic curves in cryptography*, London Mathematical Society Lecture Note Series, vol. 265, Cambridge University Press, 1999. MR1771549 (2001i:94048)
12. Reinier Bröker, *A p-adic algorithm to compute the Hilbert class polynomial*, Mathematics of Computation **77** (2008), 2417–2435. MR2429891 (2009j:11093)
13. Reinier Bröker and Peter Stevenhagen, *Efficient CM-constructions of elliptic curves over finite fields*, Mathematics of Computation **76** (2007), 2161–2179. MR2336289 (2008i:11077)
14. Johannes Buchmann and Arthur Schmidt, *Computing the structure of a finite abelian group*, Mathematics of Computation **74** (2005), 2017–2026. MR2164109 (2006c:20108)
15. Johannes Buchmann and Ulrich Vollmer, *Binary quadratic forms: An algorithmic approach*, Algorithms and Computations in Mathematics, vol. 20, Springer, 2007. MR2300780 (2008b:11046)
16. Frank Celler and C. R. Leedham-Green, *Calculating the order of an invertible matrix*, Groups and Computation II, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 28, American Mathematical Society, 1997, pp. 55–60. MR1444130 (98g:20001)
17. Jinhui Chao, Osamu Nakamura, Kohji Sobataka, and Shigeo Tsujii, *Construction of secure elliptic cryptosystems using CM tests and liftings*, Advances in Cryptology–ASIACRYPT'98, Lecture Notes in Computer Science, vol. 1514, Springer, 1998, pp. 95–109. MR1727916
18. Henri Cohen, *A course in computational algebraic number theory*, Springer, 1996. MR1228206 (94i:11105)
19. ______, *High precision computation of Hardy-Littlewood constants*, 1999, available at `http://www.math.u-bordeaux.fr/~cohen/hardylw.dvi`.
20. Henri Cohen, Gerhard Frey, Roberto Avanzi, Christophe Doche, Tanja Lange, Kim Nguyen, and Frederik Vercauteren, *Handbook of elliptic and hyperelliptic curve cryptography*, Chapman and Hall, 2006. MR2162716 (2007f:14020)
21. Paula Cohen, *On the coefficients of the transformation polynomials for the elliptic modular function*, Math. Proc. of the Cambridge Philosophical Society **95** (1984), 389–402. MR755826 (85k:11020)

22. Jean-Marc Couveignes and Thierry Henocq, *Action of modular correspondences around CM points*, Algorithmic Number Theory Symposium–ANTS V (C. Fieker and D. R. Kohel, eds.), Lecture Notes in Computer Science, vol. 2369, Springer-Verlag, 2002, pp. 234–243. MR2041087 (2005b:11077)
23. David A. Cox, *Primes of the form $x^2 + ny^2$: Fermat, class field theory, and complex multiplication*, John Wiley and Sons, 1989. MR1028322 (90m:11016)
24. Richard Crandall and Carl Pomerance, *Prime numbers: A computational perspective*, second ed., Springer, 2005. MR2156291 (2006a:11005)
25. John E. Cremona and Andrew V. Sutherland, *On a theorem of Mestre and Schoof*, Journal de Théorie des Nombres de Bordeaux (2009), to appear, `http://arxiv.org/abs/0901.0120`.
26. Max Deuring, *Die Typen der Multiplikatorenringe elliptischer Funktionenkörper*, Abh. Math. Sem. Hansischen Univ. **14** (1941), 197–272. MR0005125 (3:104f)
27. Andreas Enge, *The complexity of class polynomial computation via floating point approximations*, Mathematics of Computation **78** (2009), 1089–1107. MR2476572
28. ———, *Computing modular polynomials in quasi-linear time*, Mathematics of Computation **78** (2009), 1809–1824. MR2501077
29. 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. MR2041091 (2005c:11077)
30. David Freeman, *Constructing pairing-friendly elliptic curves with embedding degree* 10, Algorithmic Number Theory Symposium–ANTS VII (F. Hess, S. Pauli, and M. Pohst, eds.), Lecture Notes in Computer Science, vol. 4076, Springer, 2006, pp. 452–465. MR2282942 (2008b:14042)
31. David Freeman, Michael Scott, and Edlyn Teske, *A taxonomy of pairing-friendly elliptic curves*, Journal of Cryptology (2009), DOI: 10.1007/s00145-009-9048-z, to appear in print.
32. Ernst-Ulrich Gekeler, *The distribution of group structures on elliptic curves over finite prime fields*, Documenta Mathematica **11** (2006), 119–142. MR2226271 (2007b:11143)
33. Torbjörn Granlund et al., *GNU multiple precision arithmetic library*, September 2008, version 4.2.4, available at `http://gmplib.org/`.
34. James L. Hafner and Kevin S. McCurley, *A rigorous subexponential algorithm for computation of class groups*, Journal of the American Mathematical Society **2** (1989), no. 4, 837–850. MR1002631 (91f:11090)
35. David Harvey, *Faster polynomial multiplication via multipoint Kronecker substitution*, Journal of Symbolic Computation **44** (2009), no. 10, 1502-1510. MR2543433
36. ———, *zn_poly: A library for polynomial arithmetic*, 2008, version 0.9, `http://cims.nyu.edu/~harvey/zn_poly`.
37. Derek F. Holt, Bettina Eick, and Eamonn A. O'Brien, *Handbook of computational group theory*, CRC Press, 2005. MR2129747 (2006f:20001)
38. Everett W. Howe, *On the group orders of elliptic curves over finite fields*, Compositio Mathematica **85** (1993), 229–247. MR1204781 (94a:11089)
39. Dale Husemöller, *Elliptic curves*, Springer-Verlag, 1987. MR868861 (88h:11039)
40. Michael J. Jacobson, Jr., S. Ramachandran, and Hugh C. Williams, *Numerical results on class groups of imaginary quadratic fields*, Algorithmic Number Theory Symposium–ANTS VII (F. Hess, S. Pauli, and M. Pohst, eds.), Lecture Notes in Computer Science, vol. 4076, Springer, 2006, pp. 87–101. MR2282917 (2007j:11178)
41. ———, *Supplementary tables for "Numerical results on class groups of imaginary quadratic fields"*, 2006, available at `http://www.math.tu-berlin.de/~kant/ants/Proceedings/ramachandran-74/ramachandran-74-tables.pdf`.
42. Daeyeol Joen and Chang Heon Kim, *On the arithmetic of certain modular curves*, 2006, `http://arxiv.org/abs/math/0607611v1`.
43. Ezekiel J. Kachisa, Edward F. Schaefer, and Michael Scott, *Constructing brezingweng pairing friendly elliptic curves using elements in the cyclotomic field*, Pairing-Based Cryptography-Pairing 2008, Lecture Notes in Computer Science, vol. 5209, Springer, 2008, p. 126135.
44. Koray Karabina and Edlyn Teske, *On prime-order elliptic curves with embedding degrees* $k = 3, 4,$ *and* 6, 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. 102–117. MR2467839

45. Kiran S. Kedlaya and Andrew V. Sutherland, *Computing L-series of hyperelliptic curves*, 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. 312–326. MR2467855
46. David Kohel, *Endomorphism rings of elliptic curves over finite fields*, Ph.D. thesis, University of California at Berkeley, 1996.
47. Daniel Sion Kubert, *Universal bounds on the torsion of elliptic curves*, Proceedings of the London Mathematical Society **33** (1976), 193–237. MR0434947 (55:7910)
48. J. C. Lagarias and A. M. Odlyzko, *Effective versions of the Chebotarev density theorem*, Algebraic number fields: L-functions and Galois properties (Proc. Sympos., Univ. Durham, Duram, 1975), Academic Press, 1977, pp. 409–464. MR0447191 (56:5506)
49. Serge Lang, *Elliptic functions*, second ed., Springer-Verlag, 1987. MR890960 (88c:11028)
50. Georg-Johann Lay and Horst G. Zimmer, *Constructing elliptic curves with given group order over large finite fields*, Algorithmic Number Theory Symposium–ANTS I (L. M. Adleman and M.-D. Huang, eds.), Lecture Notes in Computer Science, vol. 877, 1994, pp. 250–263. MR1322728 (96a:11054)
51. Hendrik W. Lenstra, Jr., *Factoring integers with elliptic curves*, Annals of Mathematics **126** (1987), 649–673. MR916721 (89g:11125)
52. J. E. Littlewood, *On the class-number of the corpus $P(\sqrt{-k})$*, Proceedings of the London Mathematical Society **27** (1928), 358–372.
53. J. Miret, R. Moreno, A. Rio, and M. Valls, *Determining the 2-sylow subgroup of an elliptic curve over a finite field*, Mathematics of Computation **74** (2004), no. 249, 411–427. MR2085900 (2005d:11090)
54. J. Miret, R. Moreno, D. Sadornil, J. Tena, and M. Valls, *Computing the height of volcanoes of l-isogenies of elliptic curves over finite fields*, Applied Mathematics and Computation **196** (2008), no. 1, 67–76. MR2382590 (2008m:11122)
55. J. B. Rosser and L. Schoenfeld, *Approximate formulas for some functions of prime numbers*, Illinois Journal of Mathematics **6** (1962), 64–94. MR0137689 (25:1139)
56. Karl Rubin and Alice Silverberg, *Choosing the correct elliptic curve in the CM method*, Mathematics of Computation **79** (2010), no. 269, 545–561. MR2552240
57. A. Schönhage and V. Strassen, *Schnelle Multiplikation großer Zahlen*, Computing **7** (1971), 281–292. MR0292344 (45:1431)
58. René Schoof, *The exponents of the groups of points on reductions of an elliptic curve*, Arithmetic Algebraic Geometry (G. van der Geer, F. Oort, and J. Steenbrink, eds.), Birkhäuser, 1991, pp. 325–335. MR1085266 (91j:11043)
59. I. Schur, *Einige Bemerkungen zu der vorstehenden Arbeit des Herrn G. Polya: Über die Verteilung der quadratischen Reste und Nichtreste*, Nachr. Kon. Ges. Wiss. Göttingen, Math.-phys. Kl. (1918), 30–36, in Gesammelte Abhandlungen, vol. II, pp. 239–245, Springer, 1973.
60. Jean-Pierre Serre, *Complex multiplication*, Algebraic Number Theory (J.W.S. Cassels and A. Frölich, eds.), Academic Press, 1967. MR0244199 (39:5516)
61. Joseph H. Silverman, *The arithmetic of elliptic curves*, Springer, 1986. MR817210 (87g:11070)
62. ______, *Advanced topics in the arithmetic of elliptic curves*, Springer, 1999. MR1312368 (96b:11074)
63. Richard Stallman et al., *GNU compiler collection*, August 2008, version 4.3.2, available at `http://gcc.gnu.org/`.
64. Andrew V. Sutherland, *Order computations in generic groups*, Ph.D. thesis, MIT, 2007, `http://groups.csail.mit.edu/cis/theses/sutherland-phd.pdf`.
65. ______, *Constructing elliptic curves over finite fields with prescribed torsion*, 2008, `http://arxiv.org/abs/0811.0296`.
66. ______, *Discrete logarithms and structure computation in finite abelian p-groups*, Mathematics of Computation (2009), to appear, `http://arxiv.org/abs/0809.3413`.
67. Edlyn Teske, *A space efficient algorithm for group structure computation*, Mathematics of Computation **67** (1998), 1637–1663. MR1474658 (99a:11146)
68. Frederik Vercauteren, *Pairings on elliptic curves*, Identity-Based Cryptography (M. Joye and G. Neven, eds.), Cryptology and Information Security Series, vol. 2, IOS Press, 2008, pp. 13–30.

69. Ulrich Vollmer, *Invariant and discrete logarithm computation in quadratic orders*, Ph.D. thesis, Technische Universität Darmstadt, 2003.
70. Joachim von zur Gathen and Jürgen Gerhard, *Modern computer algebra*, second ed., Cambridge University Press, 2003. MR2001757 (2004g:68202)
71. Lawrence C. Washington, *Elliptic curves: Number theory and cryptography*, second ed., CRC Press, 2008. MR2404461 (2009b:11101)

Department of Mathematics, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139

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