LMS J. Comput. Math. 15 (2012) 172–204 © 2012 Author
doi:10.1112/S1461157012001015

Accelerating the CM method

Andrew V. Sutherland

Abstract

Given a prime q and a negative discriminant D, the CM method constructs an elliptic curve $E/\mathbf{F}_q$ by obtaining a root of the Hilbert class polynomial $H_D(X)$ modulo q. We consider an approach based on a decomposition of the ring class field defined by H_D, which we adapt to a CRT setting. This yields two algorithms, each of which obtains a root of H_D mod q without necessarily computing any of its coefficients. Heuristically, our approach uses asymptotically less time and space than the standard CM method for almost all D. Under the GRH, and reasonable assumptions about the size of $\log q$ relative to $|D|$, we achieve a space complexity of $O((m+n)\log q)$ bits, where $mn = h(D)$, which may be as small as $O(|D|^{1/4}\log q)$. The practical efficiency of the algorithms is demonstrated using $|D| > 10^{16}$ and $q \approx 2^{256}$, and also $|D| > 10^{15}$ and $q \approx 2^{33220}$. These examples are both an order of magnitude larger than the best previous results obtained with the CM method.

1. *Introduction*

The *CM method* is a widely used technique for constructing elliptic curves over finite fields. To illustrate, let us construct an elliptic curve $E/\mathbf{F}_q$ with exactly N points. We shall assume that q is prime, and require $t = q + 1 - N$ to be nonzero and satisfy $|t| < 2\sqrt{q}$. We may write $4q = t^2 - v^2 D$, for some nonzero integer v and negative discriminant D, and then proceed as follows.

(i) Compute the Hilbert class polynomial $H_D \in \mathbf{Z}[X]$.
(ii) Find a root x of $H_D(X)$ modulo q.

The root x is the j-invariant of an elliptic curve E with $\#E(\mathbf{F}_q) = N$; an explicit equation for E can be obtained via [37]. The endomorphism ring $\mathrm{End}(E)$ is isomorphic to the imaginary quadratic order $\mathcal{O}$ with discriminant D, and we say that E has *complex multiplication* (CM) by $\mathcal{O}$.

In principle, the CM method can construct any ordinary elliptic curve $E/\mathbf{F}_q$. In practice, it is feasible only when $|D|$ is fairly small. The main difficulty lies in step 1. The Hilbert class polynomial is notoriously large, as may be seen in Table 1.

The value $h(D)$ is the class number of D, which is the degree of H_D. The size listed is an upper bound on the total size of H_D derived from known bounds on its largest coefficient [42, Lemma 8], and is generally accurate to within ten percent. These discriminants were

Table 1. *Size of* $H_D(X)$.

$\lvert D\rvert$	$h(D)$	Size	$\lvert D\rvert$	$h(D)$	Size
10^5+4	152	152 KB	$10^{11}+4$	145981	323 GB
10^6+104	472	1.67 MB	$10^{12}+135$	465872	3.53 TB
10^7+47	1512	22.3 MB	$10^{13}+15$	1463328	38.5 TB
10^8+20	5056	239 MB	$10^{14}+4$	4658184	384 TB
10^9+15	15216	2.73 GB	$10^{15}+15$	14635920	4.45 PB
$10^{10}+47$	48720	31.4 GB	$10^{16}+135$	46275182	47.2 PB

Received 9 May 2011; revised 22 April 2012.
2010 Mathematics subject classification 11Y16 (primary), 11G15, 11G20, 14H52 (secondary).

chosen so that the ratio $h(D)/\sqrt{|D|}$ is within ten percent of its asymptotic average $(0.461559\ldots)$, so they represent typical examples.

There are at least three different ways to compute H_D: the complex analytic method [**3**, **21**, **26**], a p-adic approach [**12**, **18**], or by computing H_D modulo many small primes and applying the Chinese remainder theorem (CRT) [**1**, **7**, **15**]. Under suitable heuristic assumptions all three approaches can achieve quasi-linear running times: $O(|D| \log^c |D|)$ for some constant c. However, the $O(|D| \log^{1+\epsilon} |D|)$ space needed to compute H_D makes it difficult to apply these algorithms when $|D|$ is large. As noted in [**21**], space is typically the limiting factor.

Here we build on the CRT approach of [**42**], which gives a probabilistic (Las Vegas) algorithm to compute H_D, with an expected running time of $O(|D| \log^{5+\epsilon} |D|)$ under the generalized Riemann hypothesis (GRH), and a heuristic running time of $O(|D| \log^{3+\epsilon} |D|)$. Most critically for the CM method, it directly computes $H_D \bmod q$ without computing H_D over $\mathbf{Z}$. This yields a space complexity of $O\big(|D|^{1/2+\epsilon} \log q\big)$, allowing it to handle much larger values of $|D|$.

As a practical optimization, the CM method may use alternative class polynomials that are smaller than H_D by a large constant factor. The algorithm in [**42**] has recently been adapted to compute such class polynomials [**24**]. To simplify our presentation we focus on the Hilbert class polynomial H_D, but our results apply to all the class polynomials considered in [**24**], a feature we exploit in § 6.

When $h(D)$ is composite, a root of H_D can be obtained via a decomposition of the field extension defined by $H_D(X)$, as described in [**23**, **28**]. The algorithm in [**23**] computes integer polynomials that describe this decomposition, which can be used in place of H_D. This allows a single root-finding operation to be replaced by two root-finding operations of smaller degree, speeding up step (ii) of the CM method. We adapt this technique to a CRT setting, where we find it also accelerates step (i), which is the asymptotically dominant step as a function of $|D|$[†].

Provided $h(D)$ is sufficiently composite, we may choose a decomposition that significantly reduces the size of the coefficients in the defining polynomials. In order to do so, we derive an explicit height bound that can be efficiently computed for each of the possible decompositions available. By choosing the optimal decomposition we gain nearly a $\log |D|$ factor in the running time, on average, based on the heuristic analysis in § 5.4. This claim is supported by empirical data, and we give practical examples that achieve more than a tenfold speedup.

We are also able to improve the space complexity of the CM method.

PROPOSITION. *Assume the GRH, and fix real constants $\delta \geqslant 0$ and $\epsilon > 0$. Let $\mathcal{O}$ be an imaginary quadratic order with discriminant D and class number $h = mn$, with $m \leqslant O(|D|^{1/2-\delta})$. Let q be a prime of the form $4q = t^2 - v^2 D$, and assume $\log q = O(|D|^{\delta} \log |D|)$. An elliptic curve $E/\mathbf{F}_q$ with $\mathrm{End}(E) \cong \mathcal{O}$ can be constructed in $O(|D| \log^{6+\epsilon} |D|)$ expected time using $O((m+n) \log q)$ space.*

This is achieved by interleaving the computation of the defining polynomials modulo many small primes p with root-finding operations modulo q. If we additionally require $m = \Omega(|D|^{1/2-\gamma})$, for some $\gamma > \delta$, this yields an $O(|D|^{1/4+\gamma} \log q)$ space bound, improving the $O(|D|^{1/2+\epsilon} \log q)$ result in [**42**].

The organization of the paper is as follows. We begin with some necessary preparation in § 2, and then present two algorithms to obtain a root of the Hilbert (or other) class polynomial modulo a prime q in §§ 3 and 4. The optimization of the height bound is addressed in § 5. Finally, we present performance data in § 6, including the construction of an elliptic curve over a 256 bit prime field with $|D| > 10^{16}$, and an elliptic curve over a 10 000-digit prime field with $|D| > 10^{15}$, both of which are new records for the CM method.

[†]We assume throughout that fast probabilistic methods are used to find roots of polynomials over finite fields.

https://doi.org/10.1112/S1461157012001015 Published online by Cambridge University Press

2. *Background*

2.1. *Hilbert class polynomials*

We first recall some facts from the theory of complex multiplication, referring to [19, 34, 40] for proofs and further background. Let $\mathcal{O}$ be an imaginary quadratic order, identified by its discriminant D. The j-invariant of the lattice $\mathcal{O}$ is an algebraic integer whose minimal polynomial is the *Hilbert class polynomial* H_D. If $\mathfrak{a}$ is an invertible $\mathcal{O}$-ideal (including $\mathfrak{a} = \mathcal{O}$), then the torus $\mathbf{C}/\mathfrak{a}$ corresponds to an elliptic curve $E/\mathbf{C}$ with CM by $\mathcal{O}$, and every such curve arises in this fashion. Equivalent ideals yield isomorphic elliptic curves, and this gives a bijection between the ideal class group $\mathrm{cl}(\mathcal{O})$ and the set

$$\mathrm{Ell}_{\mathcal{O}}(\mathbf{C}) = \{j(E/\mathbf{C}) : \mathrm{End}(E) \cong \mathcal{O}\},$$

the j-invariants of the elliptic curves defined over $\mathbf{C}$ with CM by $\mathcal{O}$. We then have

$$H_D(X) = \prod_{j_i \in \mathrm{Ell}_{\mathcal{O}}(\mathbf{C})} (X - j_i). \tag{1}$$

The splitting field of H_D over $K = \mathbf{Q}(\sqrt{D})$ is the *ring class field* $K_{\mathcal{O}}$. It is an abelian extension whose Galois group is isomorphic to $\mathrm{cl}(\mathcal{O})$, via the Artin map.

This isomorphism can be made explicit via isogenies. Let $E/\mathbf{C}$ be an elliptic curve with CM by $\mathcal{O}$ and let $\mathfrak{a}$ be an invertible $\mathcal{O}$-ideal. After fixing an isomorphism $\mathrm{End}(E) \cong \mathcal{O}$, there is a uniquely determined separable isogeny whose kernel is the group of points annihilated by every endomorphism in $\mathfrak{a} \subset \mathcal{O} \cong \mathrm{End}(E)$. The image of this isogeny also has CM by $\mathcal{O}$, and this defines an action of the ideal group of $\mathcal{O}$ on the set $\mathrm{Ell}_{\mathcal{O}}(\mathbf{C})$. Principal ideals act trivially, and the induced action of the class group is regular. Thus the set $\mathrm{Ell}_{\mathcal{O}}(\mathbf{C})$ is a principal homogeneous space, a *torsor*, for the group $\mathrm{cl}(\mathcal{O})$. For a j-invariant j_i in $\mathrm{Ell}_{\mathcal{O}}(\mathbf{C})$ and an ideal class $[\mathfrak{a}]$ in $\mathrm{cl}(\mathcal{O})$, we write $[\mathfrak{a}]j_i$ to denote the image of j_i under the action of $[\mathfrak{a}]$.

If p is a prime that splits completely in $K_{\mathcal{O}}$, equivalently (for $p > 3$), a prime that satisfies the *norm equation*

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

for some nonzero integers t and v, then H_D splits completely in $\mathbf{F}_p[X]$ and its roots form the set $\mathrm{Ell}_{\mathcal{O}}(\mathbf{F}_p)$. Conversely, every ordinary (not supersingular) elliptic curve $E/\mathbf{F}_p$ has CM by some imaginary quadratic order $\mathcal{O}$ in which the Frobenius endomorphism corresponds to an element of norm p and trace t.

2.2. *Computing H_D with the CRT method*

The above theory suggests the following algorithm to compute H_D modulo a prime p that splits completely in $K_{\mathcal{O}}$.

(i) Find an elliptic curve E with j-invariant $j_1 \in \mathrm{Ell}_{\mathcal{O}}(\mathbf{F}_p)$.
(ii) Enumerate $\mathrm{Ell}_{\mathcal{O}}(\mathbf{F}_p)$ as $\{[\mathfrak{a}]j_1 : [\mathfrak{a}] \in \mathrm{cl}(\mathcal{O})\} = \{j_1, \ldots, j_h\}$.
(iii) Compute $H_D(X) \bmod p$ as $(X - j_1)(X - j_2) \ldots (X - j_h)$.

Step (i) essentially involves testing curves at random, so this algorithm is feasible only when p is fairly small, but see [42, § 3] for various ways in which this step may be optimized. Step (ii) is addressed in § 2.6. Step (iii) is just polynomial arithmetic, but this is usually the most expensive step [42, § 7.1].

Under the GRH we can we always work with primes p that are roughly the same size as $|D|$, no matter how big q is. We select a sufficiently large set of such small primes that split completely in $K_{\mathcal{O}}$, and compute $H_D \bmod p$ for each of them. Provided the product of these primes is larger than $2B$, where B bounds the coefficients of H_D, the polynomial H_D

https://doi.org/10.1112/S1461157012001015 Published online by Cambridge University Press

is uniquely determined by the Chinese Remainder Theorem (CRT). Explicit values for B are given in [22] and [42, Lemma 8].

However, to compute $H_D \bmod q$ in $O(|D|^{1/2+\epsilon}\log q)$ space, one cannot simply compute H_D over $\mathbf{Z}$ and then reduce it modulo q, since writing down H_D would already take too much space. Instead, one may use the explicit CRT (mod q) [9, 10], applying it in an online fashion to accumulate results modulo q that are updated after each computation of $H_D \bmod p$, as described in [42, § 6] and summarized in § 2.3. Here we also use the explicit CRT, but to obtain a root of $H_D \bmod q$ with a space complexity that may be as small as $O(|D|^{1/4+\epsilon}\log q)$, we must even avoid writing down $H_D \bmod q$. This requires some new techniques that are described in § 4.3.2.

2.3. *Explicit CRT*

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

Now suppose $M > 4|c|$ and let q be a prime (or any integer, in fact). Then we may apply the *explicit CRT mod q* [10, Theorem 3.1] to compute

$$c \equiv \left(\sum_i c_i a_i M_i - rM\right) \bmod q, \tag{3}$$

where r is the closest integer to $\sum_i c_i a_i/p_i$; when computing r, it suffices to approximate each $c_i a_i/p_i$ to within $1/(4n)$, by [10, Theorem 2.2].

When applying the explicit CRT to compute many values $c \bmod q$, say, the coefficients of a polynomial, one first computes the $a_i \pmod{p_i}$ and $M_i \pmod q$ using a product tree as described in [42, § 6.1]; this is CRT *precomputation* step, and it does not depend on the coefficients c. Then, as the reduced coefficient values $c_i = c \bmod p_i$ are computed for a particular p_i, the sum $\sum c_i a_i M_i \bmod q$ and an approximation to $\sum c_i a_i/p_i$ are updated for each coefficient, after which the c_i may be discarded; this is the CRT *update* step. Finally, when these sums have been updated for every prime p_i, one applies (3) to obtain $c \bmod q$ for each coefficient, using the sums $\sum c_i a_i M_i$ and the approximations $r \approx \sum c_i a_i/p_i$; this is the CRT *postcomputation* step. For further details, including explicit algorithms for each step, see [42, § 6.2]

2.4. *Assuming the GRH*

The Chebotarev density theorem guarantees that a set of primes S sufficient to compute H_D with the CRT method exists. We even have effective bounds on their size [33], but these are too large for our purpose. To obtain better bounds, we assume the GRH (for the Dedekind zeta function of $K_{\mathcal{O}}$), which implies that the primes in S are no more than a polylogarithmic factor larger than $|D|$, see [42, Lemma 3]. Having made this assumption, we are then in a position to apply other bounds that depend on some instance of the extended or generalized Riemann hypothesis, all of which we take to be included when we assume the GRH without qualification. In particular, we make frequent use of the bound $h(D) = O(|D|^{1/2}\log\log|D|)$, proven in [35], as well as the bound

$$\sum_{[\mathfrak{a}]\in \mathrm{cl}(\mathcal{O})} \frac{1}{N(\mathfrak{a})} = O(\log|D|\log\log|D|), \tag{4}$$

where $\mathfrak{a}$ is the invertible ideal of least norm in $[\mathfrak{a}]$, from [7, Lemma 2]. The sum in (4) may also be written as $\sum 1/A_i$, where (A_i, B_i, C_i) ranges over the primitive reduced binary quadratic forms $A_i x^2 + B_i xy + C_i y^2$ of discriminant $D = B_i^2 - 4A_i C_i$.

https://doi.org/10.1112/S1461157012001015 Published online by Cambridge University Press

2.5. *Decomposing the class equation*

We now describe an explicit method for 'decomposing' a polynomial via a decomposition of the field extension it defines. This is based on material in [23] and [28] that we adapt to our purpose here.

Let K be a number field, and let $P \in \mathbf{Z}[X]$ be a monic polynomial, irreducible over K, with splitting field M. We have in mind $K = \mathbf{Q}(\sqrt{D})$, $P = H_D$, and $M = K_{\mathcal{O}}$. We shall assume the action of $\mathrm{Gal}(M/K)$ is regular, equivalently, that $[M : K] = \deg P$, which holds in the case of interest.

Given a normal tower of fields $K \subset L \subset M$, we may decompose the extension M/K into extensions M/L and L/K via polynomials $U \in \mathbf{Z}(Y)[X]$ and $V \in \mathbf{Z}[X]$, where $P(x) = 0$ if and only if $U(x, y) = 0$ and $V(y) = 0$ for some $y \in L$. The polynomial $V(Y)$ defines the extension L/K, and for any root y of V, the polynomial $U(X, y)$ defines the extension M/L. In our application L will be identified as the fixed field of a given normal subgroup G of $\mathrm{Gal}(M/K)$.

Let us fix a root x of P, and let γx denote the conjugate of x under the action of $\gamma \in \mathrm{Gal}(M/K)$. Let $\beta_1, \ldots, \beta_n$ be the elements of G, and let $\alpha_1 G, \ldots, \alpha_m G$ be the cosets of G in $\mathrm{Gal}(M/K)$. We may factor P in $L[X]$ as $P = P_1, \ldots, P_m$, where

$$P_i(X) = \prod_{k=1}^{n} (X - \alpha_i \beta_k x) = \sum_{k=0}^{n} \theta_{ik} X^k. \tag{5}$$

Now let us pick a symmetric function s in $\mathbf{Z}[T_1, \ldots, T_n]$ for which each of the m values $y_i = s(\alpha_i \beta_1 x, \ldots, \alpha_i \beta_n x)$ are distinct, equivalently, for which $L = K(y_i)$; Lemma 1 below shows that this is easily achieved. We then define the polynomial

$$V(Y) = \prod_{i=1}^{m} (Y - y_i). \tag{6}$$

The coefficients of V lie in $\mathbf{Z}$, since each may be expressed as a symmetric integer polynomial in the roots of the monic polynomial $P \in \mathbf{Z}[X]$. For $0 \leqslant k \leqslant n$ let

$$W_k(Y) = \sum_{i=1}^{m} \theta_{ik} \frac{V(Y)}{(Y - y_i)}. \tag{7}$$

As with V, we have $W_k \in \mathbf{Z}[Y]$. Note that $W_k(Y)$ is the unique polynomial of degree less than m for which $W_k(y_i) = \theta_{ik} V'(y_i)$, by the Lagrange interpolation formula. This definition of the W_k is referred to as the *Hecke representation* in [23].

Finally, let

$$U(X, Y) = \frac{1}{V'(Y)} \sum_{k=0}^{n} W_k(Y) X^k. \tag{8}$$

For each root y_i of $V(Y)$ we then have $U(X, y_i) = P_i(X)$, with $V'(y_i) \neq 0$, since V has distinct roots. Each root of $U(X, y_i)$ is a root of $P(X)$, and every root of $P(X)$ may be obtained in this way. Notice that this construction does not require us to know the coefficients of P, but we must be able enumerate the G-orbits of its roots.

As noted in [28], for any given K, P, and G, there are only finitely many linear combinations of the elementary symmetric functions, up to scalar factor, that do not yield a symmetric function s suitable for the construction above. More precisely, we have the following lemma.

LEMMA 1. *Let $M = K(x)$ be a finite Galois extension of a number field K, let G be a normal subgroup of $\mathrm{Gal}(M/K)$ with order n and fixed field L, and let $m = [L : K]$. Let $\boldsymbol{x} = (x_1, \ldots, x_n)$ denote the G-orbit of x, and let $e_1, \ldots, e_n$ denote the elementary symmetric functions on n variables. There are at most $m^{n-1}(m-1)^{n-1}$ linear combinations of the form $s = e_1 + c_2 e_2 + \ldots + c_n e_n$, with $c_2, \ldots, c_n \in K$, for which $L \neq K(s(\boldsymbol{x}))$.*

https://doi.org/10.1112/S1461157012001015 Published online by Cambridge University Press

Proof. Let $z_k = e_k(\boldsymbol{x})$ and $L_k = K(z_1, \ldots, z_k)$. We have $L_1 \subset \ldots \subset L_n = L$, where the last equality follows from the fact that x is a root of the monic polynomial $X^n + \sum_{k=1}^{n}(-1)^k e_k X^{n-k}$ in $L_n[X]$, since $[L(x) : L] = n$. From the proof of the primitive element theorem in [44, § 6.10], we know that $K(z_1, z_2) = K(z_1 + c_2 z_2)$ for all $c_2 \in K$ not of the form $(u_i - u_1)/(v_j - v_1)$, with $j > 1$, where $z_1 = u_1, u_2, \ldots, u_r$ are conjugates in L/K, and $z_2 = v_1, v_2, \ldots, v_s$ are conjugates in L/K. Since $r, s \leqslant m$, there can be at most $m(m-1)$ such c_2. The same argument shows that $K(z_1 + c_2 z_2 + \ldots + c_{k-1} z_{k-1}, z_k) = K(z_1 + c_2 z_2 + \ldots + c_k z_k)$ for all but at most $m(m-1)$ values of $c_k \in K$, and the lemma follows. □

In the construction above, if $L = K(s(\boldsymbol{x}))$ holds for any G-orbit $\boldsymbol{x}$, then it necessarily holds for every G-orbit contained in the same $\mathrm{Gal}(M/K)$-orbit, and in this case the s-images y_i of these G-orbits are distinct, since they are $\mathrm{Gal}(L/K)$-conjugates. To find such an s explicitly, we pick random integer coefficients $c_2, \ldots, c_n$ uniformly distributed over $[0, 2m^2 - 1]$, and let $s = e_1 + c_2 e_2 + \ldots + c_k e_k$. By Lemma 1, we then have $L = K(s(\boldsymbol{x}))$ with probability at least $1 - 2^{1-n}$. In the event that $L \neq K(s(\boldsymbol{x}))$, we simply pick a new set of random coefficients and repeat until we succeed, yielding a Las Vegas algorithm. In the trivial case, $n = 1$ and we succeed on the first try with $s = e_1$; for $n > 1$ we expect to succeed with at most $1/(1 - 2^{1-n}) \leqslant 2$ attempts, on average.

2.6. *Orbit enumeration*

Recall that $\mathrm{Ell}_{\mathcal{O}}(\mathbf{F}_p)$ is a torsor for $\mathrm{cl}(\mathcal{O})$. Each subgroup G of $\mathrm{cl}(\mathcal{O})$ partitions $\mathrm{Ell}_{\mathcal{O}}(\mathbf{F}_p)$ into G-orbits that correspond to cosets of G. Given $j_1 \in \mathrm{Ell}_{\mathcal{O}}(\mathbf{F}_p)$, the set $\mathrm{Ell}_{\mathcal{O}}(\mathbf{F}_p)$ may be enumerated via [42, Algorithm 1.3], but to correctly identify its G-orbits we require some further refinements.

As in [42, § 5], we represent $\mathrm{cl}(\mathcal{O})$ using a polycyclic presentation defined by a sequence of ideals $\mathfrak{l}_1, \ldots, \mathfrak{l}_k$ of prime norm whose classes generate $\mathrm{cl}(\mathcal{O})$. The *relative order* of $[\mathfrak{l}_i]$ is the least positive integer r_i for which $[\mathfrak{l}_i^{r_i}] \in \langle [\mathfrak{l}_1], \ldots, [\mathfrak{l}_{i-1}] \rangle$. Every element $[\mathfrak{a}]$ of $\mathrm{cl}(\mathcal{O})$ may be uniquely written in the form

$$[\mathfrak{a}] = [\mathfrak{l}_1^{e_1}] \ldots [\mathfrak{l}_k^{e_k}],$$

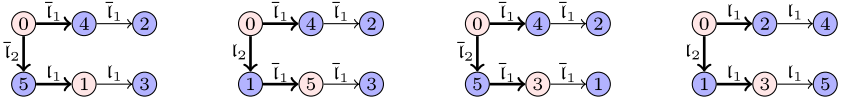
with $0 \leqslant e_i < r_i$. Having fixed a polycyclic presentation, we may enumerate $\mathrm{cl}(\mathcal{O})$ by enumerating the corresponding exponent vectors $(e_1, \ldots, e_k)$. Moreover, for any subgroup G of $\mathrm{cl}(\mathcal{O})$, we can easily identify the exponent vectors corresponding to each coset of G. This allows us to distinguish the G-orbits of $\mathrm{Ell}_{\mathcal{O}}(\mathbf{F}_p)$, provided that we can unambiguously compute the actions of $[\mathfrak{l}_1], \ldots, [\mathfrak{l}_k]$.

Let $j_1 = j(E_1)$ be an element of $\in \mathrm{Ell}_{\mathcal{O}}(\mathbf{F}_p)$ and let $\mathfrak{l}$ be an invertible $\mathcal{O}$-ideal of prime norm $\ell \neq p$. Then $j_2 = [\mathfrak{l}]j_1$ is the j-invariant of an elliptic curve E_2 that is ℓ-isogenous to E_1. We may obtain $j_2 \in \mathbf{F}_p$ as a root of $\phi(X) = \Phi_\ell(j_1, X)$, where $\Phi_N \in \mathbf{Z}[X, Y]$ is the *classical modular polynomial* [46, § 69] that parameterizes cyclic isogenies of degree N. When $[\mathfrak{l}]$ has order 2, there is just one distinct root of $\phi(X)$ that lies in $\mathrm{Ell}_{\mathcal{O}}(\mathbf{F}_p)$, but in general there will be two: $[\mathfrak{l}]j_1$ and $[\bar{\mathfrak{l}}]j_1$. Typically these are the only roots of $\phi(X)$ in $\mathbf{F}_p$, and when they are not, they may be distinguished as the roots that lie on the surface of an ℓ-volcano, see [42, § 4] or [32]. The only ambiguity lies in distinguishing the actions of $[\mathfrak{l}]$ and $[\bar{\mathfrak{l}}]$, which correspond to the two directions we may walk along the cycle of ℓ-isogenies on the surface.

To enumerate $\mathrm{Ell}_{\mathcal{O}}(\mathbf{F}_p)$ it is not necessary to distinguish these directions, as shown in [42, Proposition 5]. However, it *is* necessary, in general, if we wish to identify the G-orbits of $\mathrm{Ell}_{\mathcal{O}}(\mathbf{F}_p)$ in this enumeration. To see why, let $\mathrm{cl}(\mathcal{O})$ be a cyclic group of order 6 generated by $[\mathfrak{a}]$, and suppose our polycyclic presentation has $[\mathfrak{l}_1] = [\mathfrak{a}^2]$ with $r_1 = 3$, and $[\mathfrak{l}_2] = [\mathfrak{a}]$ with $r_2 = 2$. Below are four of the eight possible enumerations of $\mathrm{Ell}_{\mathcal{O}}(\mathbf{F}_p)$ that might be produced

https://doi.org/10.1112/S1461157012001015 Published online by Cambridge University Press

by [42, Algorithm 1.3] in this scenario.



Each node corresponds to a j-invariant $[\mathfrak{a}^e]j_1$ and is labeled by the exponent e. The arrows indicate the action of the ideal that appears in the label, and bold arrows indicate where an arbitrary choice was made. The two lightly shaded nodes in each configuration correspond to the expected positions of the elements of $\mathrm{Ell}_{\mathcal{O}}(\mathbf{F}_p)$ corresponding to the subgroup $G = \langle \mathfrak{a}^3 \rangle$ of order 2. The two configurations on the right yield a correct enumeration of each G-orbit, but the two on the left do not.

Remark 1. Of course we could have used a polycyclic presentation with $[\mathfrak{l}_1] = [\mathfrak{a}]$ and $r_1 = 6$ in this example, but unless $[\mathfrak{a}]$ happens to contain the invertible $\mathcal{O}$-ideal of least prime norm, this is suboptimal, since the cost of computing the action of $[\mathfrak{l}]$ increases quadratically with the norm of $\mathfrak{l}$. The optimal polycyclic presentation rarely has one generator, even when $\mathrm{cl}(\mathcal{O})$ is cyclic.

We now consider ways to enumerate $\mathrm{Ell}_{\mathcal{O}}(\mathbf{F}_p)$ that allow us to identify its G-orbits. First, the GCD technique of [24, § 2.3] may be used to consistently choose $\mathfrak{l}_i$ or $\bar{\mathfrak{l}}_i$ each time a new path of ℓ_i-isogenies is begun (this actually speeds up the enumeration, so it should be done in any case). We are then left with just k choices, one for each $\mathfrak{l}_i$. To consistently orient these choices we may either: (a) use auxiliary relations $[\mathfrak{a}_i] = [\mathfrak{l}_1 \dots \mathfrak{l}_i]$, where $\mathfrak{a}_i$ has (small) prime norm different from $\ell_1, \dots, \ell_i$, as described in [24, § 4.3]; or (b) compute the action of Frobenius on the kernel of the two isogenies, as discussed in [13, § 4] and the references therein. Option (a) is fast and easy to implement, but for the best space complexity we should use (b), since it does not require us to store the entire enumeration. A minor drawback to option (b) is that it requires $\ell_i \nmid v$, where v is as in the norm equation (2).

However, in many cases neither (a) nor (b) is necessary. If G is of the form

$$G = \langle [\mathfrak{l}_1], \dots, [\mathfrak{l}_{d-1}], [\mathfrak{l}_d^e] \rangle \tag{9}$$

for some $d \leqslant k$ and $e | r_d$, then it follows from the proof of [42, Proposition 5] that any enumeration of $\mathrm{Ell}_{\mathcal{O}}(\mathbf{F}_p)$ output by [42, Algorithm 1.3] also gives a correct enumeration of the G-orbits of $\mathrm{Ell}_{\mathcal{O}}(\mathbf{F}_p)$. In the context of the CM method, we are free to choose G, and we can always choose one that satisfies (9). This may limit our choices for G, but in practice this restriction is usually not a burden.

3. *A first algorithm*

Our first algorithm is a direct implementation of the theory presented in § 2, which can significantly accelerate the CM method in the typical case where the class number $h(D)$ is composite. At this stage we shall not be concerned with improving space complexity; this will be the focus of our second algorithm. Both algorithms are probabilistic (of Las Vegas type).

As above, $\mathcal{O}$ is an imaginary quadratic order with discriminant D, and we shall assume $h = h(D) > 1$. Let $\mathcal{P}_D$ be the set of primes that split completely in $K_{\mathcal{O}}$, equivalently, primes that satisfy (2). Given D and a prime $q \in \mathcal{P}_D$ we wish to obtain a root of the Hilbert class polynomial H_D over $\mathbf{F}_q$. Such a root is the j-invariant of an elliptic curve $E/\mathbf{F}_q$ with CM by $\mathcal{O}$, as required by the CM method.

The first step is to choose a subgroup G of $\mathrm{cl}(O)$, which determines $n = |G|$ and $m = h/n$. The quantities θ_{ik} and y_i, $V(Y)$, $W_k(Y)$, and $U(X, Y)$, are then defined as in § 2.5, with $1 \leqslant i \leqslant m$ and $0 \leqslant k < n$ (we do not need $k = n$).

https://doi.org/10.1112/S1461157012001015 Published online by Cambridge University Press

The y_i (and therefore $V(Y)$) depend on the choice of a function $s \in \mathbf{Z}[X_1, \ldots, X_n]$ that is a randomly chosen linear combination of the elementary symmetric functions $e_1, \ldots, e_n$, as described at the end of § 5. In the unlikely event that we pick a bad s, we may find that V is a perfect power. In this case the algorithm simply repeats the computation with a new choice of s.

We also require a bound B on the coefficients of V and W_k. The computation of B is addressed in § 5. We now give the algorithm.

ALGORITHM 1. Given D and $q \in \mathcal{P}_D$, find a root x of H_D in $\mathbf{F}_q$ as follows:

1. Select a subgroup G of $\mathrm{cl}(O)$ with $2|G|^2 \leqslant q$ and let $n = |G|$.
2. Generate random integers $c_2, \ldots, c_n$ uniformly distributed over $[0, 2m^2 - 1]$ and set $s = e_1 + c_2 e_2 + \ldots + c_n e_n$.
3. Compute a bound B as described in § 5.
4. As in steps 1–3 of [**42**, Alg. 2], use B to select $S \subset \mathcal{P}_D$, compute a polycyclic presentation Γ for $\mathrm{cl}(O)$, and perform CRT precomputation (see § 2.3).
5. For each $p \in S$:
   (a) Find $j_1 \in \mathrm{Ell}_{\mathcal{O}}(\mathbf{F}_p)$ as in [**42**, Alg. 1].
   (b) Enumerate the G-orbits G_i of $\mathrm{Ell}_{\mathcal{O}}(\mathbf{F}_p)$ using j_1 and Γ.
   (c) Compute the θ_{ik} and the $y_i \bmod p$ (using s), as described in § 2.5
   (d) Compute V and the $W_k \bmod p$.
   (e) Update CRT data for the coefficients of V and the W_k.
6. Perform CRT postcomputation to obtain V and the $W_k \bmod q$.
7. Working in $\mathbf{F}_q$, find a root y of V that is not a root of V'.
   If no such root exists then return to step 2.
8. Compute $U_y(X) = U(X, y) \bmod q$ and output a root x of U_y in $\mathbf{F}_q$.

It follows from § 2.5 that the output value x is a root of H_D in $\mathbf{F}_q$, thus the algorithm is correct. Lemma 1 guarantees that it always terminates, and that its expected running time is no more than twice the expected time when the first choice of s works (this factor approaches 1 as n increases).

REMARK 2. For practical implementation one may prefer to fix $s = e_1$ (or $s = -e_1$) and instead change the choice of G if $s = e_1$ does not work. Using $s = e_1$ simplifies the implementation and can reduce the bound B significantly. Empirically, for large values of $|D|$ and q, using $s = e_1$ is very likely to work with every choice of G. Alternatively, one may start with $s = e_1$ and then switch to a random s if necessary. In all our examples, including all the computations in § 6, and in the heuristic analysis of § 5.4, we use $s = \pm e_1$, but our mathematical results (Propositions 1–3) all assume a random s, as specified in Algorithm 1.

We note three immediate generalizations of Algorithm 1. First, other class invariants may be treated with suitable modifications to step 2, see [**24**]. Second, it is not necessary for q to be prime; q may be a prime power q_0^e satisfying $4q = t^2 - v^2 D$ with $t \not\equiv 0 \bmod q_0$. One then computes V and the $W_k \bmod q_0$, and performs the root-finding operations in $\mathbf{F}_q$. Third, we can compute V and the W_k over $\mathbf{Z}$ using the standard CRT: replace q by the product of the primes in S and lift the results of step 4 to $\mathbf{Z}$. Steps 7 and 8 may then later be applied to any q that splits completely in the ring class field.

3.1. *Example*

Let us find a root of $H_D \bmod q$ using Algorithm 1, with $D = -971$ and $q = 1029167$. The class group is cyclic of order 15, and the optimal polycyclic presentation[†] has norms $\ell_1 = 3$

[†]The symmetry of the ℓ_i and r_i in this small example is entirely coincidental.

https://doi.org/10.1112/S1461157012001015 Published online by Cambridge University Press

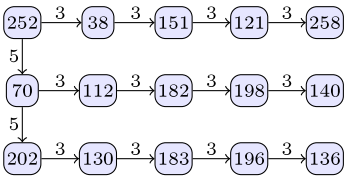
and $\ell_2 = 5$ and relative orders $r_1 = 5$ and $r_2 = 3$. We choose G to be the subgroup of order $n = 5$, which is conveniently of the form (9), so we need not distinguish directions when enumerating $\mathrm{Ell}_{\mathcal{O}}(\mathbf{F}_p)$. For convenience we set $s = -e_1$, so that $y_i = \theta_{i,n-1}$).

Computing the bound B as in § 5, we have $b = \log_2 B \approx 340$ bits, and select a set of primes in $\mathcal{P}_D$ whose product exceeds $4B$. As described in [**42**, § 3], we choose primes that optimize the search for $j_1 \in \mathrm{Ell}_{\mathcal{O}}(\mathbf{F}_p)$, obtaining a set of 27 primes:

$$S = \{263, 353, 1871, \ldots, 38677, 43237, 62873\}.$$

We then precompute parameters for the explicit CRT (mod q), using [**42**, Algorithm 2.3].

For $p = 263$ we find $j_1 = 252$, and enumerate $\mathrm{Ell}_{\mathcal{O}}(\mathbf{F}_p)$ from j_1 as,



The horizontal arrows denote 3-isogenies and the vertical arrows are 5-isogenies. The three G-orbits of $\mathrm{Ell}_{\mathcal{O}}(\mathbf{F}_p)$ are $\{252, 38, 151, 121, 258\}$, $\{70, 112, 182, 198, 140\}$, and $\{202, 130, 183, 196, 136\}$, corresponding to the rows in the diagram above. Note that these orbits do not depend on the choice of direction made at the start of each line of isogenies, nor do they depend on the choice of j_1.

Continuing with $p = 263$, we compute the products

$$\begin{aligned}
P_1(X) &\equiv_p (X - 252)(X - 38)(X - 151)(X - 121)(X - 258),\\
P_2(X) &\equiv_p (X - 70)(X - 112)(X - 182)(X - 198)(X - 140),\\
P_3(X) &\equiv_p (X - 202)(X - 130)(X - 183)(X - 196)(X - 136).
\end{aligned}$$

Each θ_{ik} is obtained as the coefficient of X^k in the polynomial P_i:

$$\begin{aligned}
P_1(X) &\equiv_p X^5 + 232X^4 + 32X^3 + 159X^2 + 208X + 158,\\
P_2(X) &\equiv_p X^5 + 87X^4 + 252X^3 + 139X^2 + 103X + 21,\\
P_3(X) &\equiv_p X^5 + 205X^4 + 86X^3 + 113X^2 + 121X + 116.
\end{aligned}$$

We then set $y_1 = \theta_{14} = 232$, $y_2 = \theta_{24} = 87$, and $y_3 = \theta_{34} = 205$. Using the values y_i and the θ_{ik}, we compute

$$\begin{aligned}
V(Y) &= (Y - y_1)(Y - y_2)(Y - y_3) \equiv_p Y^3 + 2Y^2 + 104\,Y + 59,\\
W_0(Y) &= \sum \theta_{i0} V(Y)/(Y - y_i) \equiv_p 32Y^2 + 259\,Y + 152,\\
W_1(Y) &= \sum \theta_{i1} V(Y)/(Y - y_i) \equiv_p 169\,Y^2 + 41\,Y + 153,\\
W_2(Y) &= \sum \theta_{i2} V(Y)/(Y - y_i) \equiv_p 148\,Y^2 + 117\,Y + 277,\\
W_3(Y) &= \sum \theta_{i3} V(Y)/(Y - y_i) \equiv_p 107\,Y^2 + 115\,Y + 244.
\end{aligned}$$

We do not need W_4 because $y_i = \theta_{i,n-1}$ implies $W_{n-1}(y_i) = y_i V'(y_i)$, hence the coefficient of X^{n-1} in $U(X, y_i)$ is just y_i. We complete our work for $p = 263$ by updating the CRT coefficient data (mod q) for the 15 nontrivial coefficients above, using [**42**, Algorithm 2.4]. The same procedure is then applied for each $p \in S$.

https://doi.org/10.1112/S1461157012001015 Published online by Cambridge University Press

Having processed all the primes in S, a small postcomputation step [42, Algorithm 2.5] yields V and the W_k modulo q:

$$\begin{aligned} V(Y) &\equiv_q Y^3 + 947907\,Y^2 + 829791\,Y + 760884,\\ W_0(Y) &\equiv_q 975377\,Y^2 + 130975\,Y + 363724,\\ W_1(Y) &\equiv_q 240332\,Y^2 + 135971\,Y + 616131,\\ W_2(Y) &\equiv_q 126738\,Y^2 + 479879\,Y + 908580,\\ W_3(Y) &\equiv_q 340801\,Y^2 + 1000285\,Y + 68659. \end{aligned}$$

The roots of V mod q are $y_1 = 336976$, $y_2 = 898530$, and $y_3 = 904088$, none of which are roots of V' mod q. Using $y = y_1$ we compute

$$\begin{aligned} U(X,y) &= X^5 + yX^4 + \frac{1}{V'(y)}(W_3(y)X^3 + W_2(y)X^2 + W_1(y)X + W_0(y)),\\ &\equiv_q X^5 + 336976X^4 + 556976X^3 + 849678X^2 + 363260X + 95575, \end{aligned}$$

and we then find that $x = 590272$ is a root of $U(X, y)$, and hence of H_D, modulo q.

This completes the execution of Algorithm 1 on the inputs $D = -971$ and $q = 1029167$. If we now set $k = x/(1728 - x) \equiv_q 638472$, then the elliptic curve $E/\mathbf{F}_q$ defined by

$$Y^2 = X^3 + 3kX + 2k \equiv_q X^3 + 886249X + 247777$$

has complex multiplication by the quadratic order with discriminant -971.

3.2. *Complexity*

The running time of Algorithm 1 has have four principal components. Let us define T_{find}, T_{enum}, and T_{build} (respectively) as the average expected time, over $p \in S$, to: find $j_1 \in \text{Ell}_{\mathcal{O}}(\mathbf{F}_p)$ (step 5a), enumerate the G-orbits of $\text{Ell}_{\mathcal{O}}(\mathbf{F}_p)$ (step 5b), and build the polynomials V and W_k modulo p (steps 5c and 5d). Additionally, let T_{root} be the expected time to find a root y of V mod q and to compute and find a root of $U(X, y)$ mod q (steps 7 and 8). As shown in [42], the cost of the precomputation in step 4 and the total cost of all CRT computations are negligible, as are steps 1–3. Thus the total expected running time is

$$O(|S|(T_{\text{find}} + T_{\text{enum}} + T_{\text{build}}) + T_{\text{root}}). \tag{10}$$

When G is trivial, or equal to $\text{cl}(\mathcal{O})$, Algorithm 1 reduces to the standard CM method, computing H_D mod q as in [42, Algorithm 2]. In this case, under the GRH we have the following bounds:

$$\begin{aligned} |S| &= O(|D|^{1/2} \log\log|D|),\\ T_{\text{find}} &= O(h \log^{5+\epsilon} h),\\ T_{\text{enum}} &= O(h \log^{5+\epsilon} h),\\ T_{\text{build}} &= O(h \log^{3+\epsilon} h),\\ T_{\text{root}} &= O(h \log h \log^{2+\epsilon} q). \end{aligned}$$

The first four bounds are proved in [42, Lemmas 7 and 8].

The last bound is based on the standard probabilistic root-finding algorithm of [8, § 7], using fast arithmetic in $\mathbf{F}_p[x]$. With Kronecker substitution, multiplying two polynomials of degree d in $\mathbf{F}_q[x]$ uses $O(\mathsf{M}(d \log q))$ bit operations, and this yields an $O(\mathsf{M}(d \log q) \log q)$ bound on the expected time to find a single root of a polynomial in $\mathbf{F}_q[x]$ of degree d. Here $\mathsf{M}(n)$ denotes the time to multiply two n-bit integers [45, Definition 8.26], which we assume to be superlinear, and which satisfies the Schönhage–Strassen bound $\mathsf{M}(n) = O(n \log n \log\log n)$; see [39].

https://doi.org/10.1112/S1461157012001015 Published online by Cambridge University Press

When $n = |G|$ properly divides the class number h, the times for T_{build} and T_{root} may change; these are analyzed below. The times T_{find} and T_{enum} are independent of the choice of G, as is the space complexity. Depending on the bound B, the size of S may also be reduced. This can lead to a substantial practical improvement, depending on the choice of G and the value of s, but we defer this issue to § 5. For the moment we simply note that the bound on $|S|$ above holds for any choice of G and for every s.

We now consider the time T_{build}. Computing the θ_{ik} in step 5c via (5) involves building m polynomials P_i of degree n in $\mathbf{F}_p[X]$ as products of their linear factors. Let $\mathsf{M}(n)$ denote the time to multiply two n-bit integers [**45**, Definition 8.26], which we assume to be superlinear. With Kronecker substitution, multiplying two polynomials of degree n in $\mathbf{F}_p[x]$ uses $O(\mathsf{M}(n \log p))$ bit operations. Using a product tree for each P_i yields the bound[†]

$$O(m\mathsf{M}(n \log p) \log n) \subset O(h \log^2 n \log^{1+\epsilon} p) \tag{11}$$

on the cost of computing the θ_{ik}. Here we have used $4p > |D| > h \geqslant n$ and the $O(n \log n \log\log n)$ bound [**39**] for $\mathsf{M}(n)$, which we note applies to the algorithms used in our implementation. The cost of computing the y_i is also dominated by the time to build m polynomials of degree n as products of their linear factors: for each G-orbit G_i we compute the polynomial $\prod_{j \in G_i}(X - j) \in \mathbf{F}_p[X]$ whose coefficients are the values of the symmetric functions $e_1, \ldots, e_n$ on G_i (up to a sign), from which we compute the linear combination $y_i = s$.

The cost of building V as a product of its linear factors is $O(\mathsf{M}(m \log p) \log m)$, which is dominated by the cost of computing the n polynomials W_k as linear combinations of $V(Y)/(Y - y_i)$ with coefficients θ_{ik}. Using a recursive algorithm to compute each W_k as in [**45**, Algorithm 10.9], we obtain a total cost of

$$O(n\mathsf{M}(m \log p) \log m) \subset O(h \log^2 m \log^{1+\epsilon} p) \tag{12}$$

for each iteration of step 5d. The sum of (11) and (12) is $O(h \log^2 h \log^{1+\epsilon} p)$, essentially the same as the cost of building $H_D \bmod p$ from its linear factors. There is an improvement in the implicit constants, but asymptotically we gain at most a factor of 2 in the time T_{build}.

We may gain much more in the time T_{root}. The total expected time for steps 7 and 8 of Algorithm 1 is bounded by

$$T_{\text{root}} = O(\mathsf{M}(m \log q) \log q + h\mathsf{M}(\log q) + \mathsf{M}(n \log q) \log q). \tag{13}$$

The three terms in (13) reflect the time to: (a) find root y of $V \bmod q$ that is not a root of $V' \bmod q$ (b) compute $U_y(X) = U(X, y) \bmod q$ and (c) find a root of $U_y \bmod q$. In (a), any common roots of V and V' are first removed from V via repeated division by the GCD, which takes negligible time. We may bound (13) by

$$T_{\text{root}} = O(h \log^{1+\epsilon} q + (m + n) \log h \log^{2+\epsilon} q), \tag{14}$$

improving T_{root} by a factor of $\min(\log h \log q, mn/(m + n))$ compared to the time to find a root of $H_D \bmod q$. This can reduce the cost of root-finding dramatically, as may be seen in § 6.

4. *A second algorithm*

Algorithm 1 obtains a root of $H_D \bmod q$ as a root of $U(X, y) \bmod q$, where y is a root of $V \bmod q$, and U is defined by

$$U(X, Y) = \frac{1}{V'(Y)} \sum_{k=0}^{n} W_k(Y) X^k. \tag{8}$$

[†]Our bounds count bit operations and hold for all constants $\epsilon > 0$ (and often for $\epsilon = o(1)$).

https://doi.org/10.1112/S1461157012001015 Published online by Cambridge University Press

To compute $U(X, y)$ we need to evaluate each W_k at $y \in \mathbf{F}_q$. We observe that it is not necessary to know the coefficients of W_k to do this, we could instead use

$$W_k(Y) = \sum_{i=1}^{m} \theta_{ik} \frac{V(Y)}{(Y - y_i)}, \tag{7}$$

provided that we know V and the θ_{ik} (which determine the y_i).

Unfortunately we do not know the θ_{ik} in $\mathbf{F}_q$. Algorithm 1 computes the θ_{ik} in $\mathbf{F}_p$, for each prime $p \in S$, but we cannot readily export this knowledge to $\mathbf{F}_q$ because the θ_{ik} do not correspond to the reductions of rational integers[†]. Indeed, the entire reason for using the polynomials V and W_k is that they have coefficients in $\mathbf{Z}$ and are thus defined over any field.

However, there is nothing to stop us from using (7) to evaluate W_k in $\mathbf{F}_p$. Given any $z \in \mathbf{Z}$, we can certainly compute $W_k(z) \bmod p$, and if we do this for sufficiently many p we can apply the explicit CRT (mod q) to obtain $W_k(z) \bmod q$.

In particular, we can apply this to a lift of $y \in \mathbf{F}_q \cong \mathbf{Z}/q\mathbf{Z}$ to $\mathbf{Z}$. Explicitly, let $\varphi = \phi\pi$, where π is the unique field isomorphism from $\mathbf{F}_q$ to $\mathbf{Z}/q\mathbf{Z}$ and ϕ maps each residue class in $\mathbf{Z}/q\mathbf{Z}$ to its unique representative in the interval $[0, q-1]$. We then have $W_k(\varphi(y)) \equiv \varphi(W_k(y)) \bmod q$.

Thus it suffices to compute $w_k = W_k(\varphi(y)) \bmod q$, and this can be accomplished by computing $w_k \bmod p$ for sufficiently many primes p. Note that while y is a root of $V \bmod q$, when we reduce $\varphi(y)$ modulo p, we should not expect to get a root of $V \bmod p$, nor do we need to; we are simply evaluating the integer polynomial $W_k(Y)$ at the integer $\varphi(y)$, modulo many primes p.

This leads to our second algorithm, which proceeds in two stages. The first stage computes $V \bmod q$ and finds a root y. The second stage computes the values $w_k \bmod q$, then computes $U(X, y) \bmod q$ and finds a root x. The second stage requires a bound on $|w_k|$, for which we may use Bmq^{m-1}, where B bounds the coefficients of the W_k. For the sake of simplicity we use a single bound for both stages (mq^{m-1} times the bound used in Algorithm 1), but in practice one may compute separate bounds for each stage (in stage 1 it is only necessary to bound the coefficients of V). In order to achieve the best space complexity, certain steps are intentionally repeated, and some may require a more careful implementation, see § 4.3 for details.

As before, the choice of $G \subset \mathrm{cl}(\mathcal{O})$ in step 1 determines $n = |G|$ and $m = h/n$, and the values y_i, θ_{ik} and W_k are defined for $1 \leqslant i \leqslant m$ and $0 \leqslant k < n$, where the y_i depend on the symmetric function s constructed in step 1.

ALGORITHM 2. Given D and $q \in \mathcal{P}_D$, compute a root x of $H_D \bmod q$ as follows.

1. Select G, generate a random s, and compute B, as in Algorithm 1, then set $B \leftarrow mq^{m-1}B$.
2. Compute a polycyclic presentation Γ for $\mathrm{cl}(\mathcal{O})$.
3. Use B to select $S \subset \mathcal{P}_D$, and perform CRT precomputation (mod q)[‡].
4. For each $p \in S$:
   (a) Find $j_1 \in \mathrm{Ell}_{\mathcal{O}}(\mathbf{F}_p)$ as in [**42**, Alg. 1], and cache $j_1(p) = j_1$.
   (b) Enumerate the G-orbits G_i of $\mathrm{Ell}_{\mathcal{O}}(\mathbf{F}_p)$ using j_1 and Γ.
   (c) Compute the y_i and $V \bmod p$.
   (d) Update CRT data for $V \bmod q$.
5. Perform CRT postcomputation to obtain $V \bmod q$.
6. Find a root y of $V \bmod q$ that is not a root of $V' \bmod q$. If no such root exists then return to step 1.
7. For each $p \in S$:
   (a) Let $j_1 = j_1(p)$ be the element of $\mathrm{Ell}_{\mathcal{O}}(\mathbf{F}_p)$ computed in step 4a.

[†]The θ_{ik} are integers of G's fixed field $L \subset K_{\mathcal{O}}$. They do not all lie in $\mathbf{Q}$ unless $G = \mathrm{cl}(\mathcal{O})$.

[‡]To optimize space this step may need to be performed in an amortized fashion, see § 4.3.

https://doi.org/10.1112/S1461157012001015 Published online by Cambridge University Press

(b) Enumerate the G-orbits G_i of $\mathrm{Ell}_{\mathcal{O}}(\mathbf{F}_p)$ using j_1 and Γ.
(c) Compute the values θ_{ik}, y_i, $\phi(y)$ and $V \bmod p$, and then use them to compute the values $w_k = W_k(\varphi(y)) \bmod p$, via the formula in (7)[†].
(d) Update CRT data for the $w_k \bmod q$.

8. Perform CRT postcomputation to obtain the $w_k \bmod q$.
9. Compute $U_y(X) = U(X, y) \bmod q$ and output a root x of $U_y \bmod q$.

Computing the w_k in step 7c via (7) is faster than computing the coefficients of W_k, by a factor of $\log^2 m$. For a suitable choice of G (with $n = |G|$ small, say polylogarithmic in h), this may improve the time complexity of the entire algorithm, as shown in § 4.2. However, it is often better to choose G to optimize the bound B, as described in § 5, which will tend to make G large.

More significantly, computing the scalars w_k rather than the polynomials W_k reduces the space complexity to $O(h \log h + (m+n)\log q)$, which may be much better than the $O(h \log q)$ space complexity of Algorithm 1 when q is large. This can even be improved to $O((m+n)\log q)$ using a more intricate implementation, but this may increase the time complexity slightly. The details are given in § 4.3, where the space complexity of Algorithm 2 is analyzed.

4.1. *Example*

Let us revisit the example of § 3.1, with $D = -971$ and $q = 1029167$. As before, the class group is cyclic of order $h = 15$, we let G be the subgroup of order $n = 5$, and set $s = -e_1$ so that $y_i = \theta_{i,n-1}$.

The first stage of Algorithm 2 proceeds as in Algorithm 1, except that our height bound is now $\log_2(mq^{m-1}B) \approx 366$ bits, so we select a slightly larger set S, with 29 primes whose product exceeds $4mq^{m-1}B$. For $p = 263$ we again find that the three G-orbits of $\mathrm{Ell}_{\mathcal{O}}(\mathbf{F}_p)$ are $\{252, 38, 151, 121, 258\}$, $\{70, 112, 182, 198, 140\}$, and $\{202, 130, 183, 196, 136\}$. But instead of computing all the θ_{ik} as in Algorithm 1, we only need to compute

$$\begin{aligned} y_1 &\equiv_p -(252 + 38 + 151 + 121 + 258) \equiv_p 232, \\ y_2 &\equiv_p -(70 + 112 + 182 + 198 + 140) \equiv_p 87, \\ y_3 &\equiv_p -(202 + 130 + 183 + 196 + 136) \equiv_p 205, \end{aligned}$$

which yields

$$V(Y) = (Y - y_1)(Y - y_2)(Y - y_3) \equiv_p Y^3 + 2\,Y^2 + 104\,Y + 59.$$

After computing $V \bmod p$ for all $p \in S$ we obtain, via the explicit CRT $(\bmod q)$,

$$V(Y) \equiv_q Y^3 + 947907\,Y^2 + 829791\,Y + 760884,$$

and we again find that $y = 336976$ is a root of V and not V'. We now lift y from $\mathbf{F}_q \cong \mathbf{Z}/q\mathbf{Z}$ to the integer $\varphi(y) = 336976$ and begin the second stage.

For $p = 263$ we recompute the three G-orbits as above, and this time we compute all of the θ_{ik}, as we did in Algorithm 1. Setting $y_1 = \theta_{14} = 49$, $y_2 = \theta_{24} = 87$, and $y_3 = \theta_{34} = 205$, we recompute $V \bmod p$ as above.

We now let z be the reduction of $\varphi(y) \bmod p$ and find that $z = 73$ (which is not a root of $V \bmod p$, as expected). We then compute

$$\begin{aligned} z_1 &= V(z)/(z - y_1) = (z - y_2)(z - y_3) \equiv_p 7, \\ z_2 &= V(z)/(z - y_2) = (z - y_1)(z - y_3) \equiv_p 211, \\ z_3 &= V(z)/(z - y_3) = (z - y_1)(z - y_2) \equiv_p 122. \end{aligned}$$

[†]It is possible to do this without explicitly computing $V \bmod p$, see § 4.1.

https://doi.org/10.1112/S1461157012001015 Published online by Cambridge University Press

The z_i can be computed in two ways: (a) evaluate $V(z)$, invert the $(z - y_i)$, and compute the z_i's as products or (b) simultaneously compute the z_i as the complements of the $(z - y_i)$ using a product tree, as in [**42**, § 6.1]. Both use $O(m)$ field operations, but (b) is faster and works even when z is a root of V mod p.

We now compute the w_k as linear combinations of the z_i with coefficients θ_{ik}:

$$\begin{aligned} w_0 &= W_0(z) = \theta_{10}z_1 + \theta_{20}z_2 + \theta_{30}z_3 \equiv_p 227, \\ w_1 &= W_1(z) = \theta_{11}z_1 + \theta_{21}z_2 + \theta_{31}z_3 \equiv_p 79, \\ w_2 &= W_0(z) = \theta_{12}z_1 + \theta_{22}z_2 + \theta_{32}z_3 \equiv_p 44, \\ w_3 &= W_0(z) = \theta_{13}z_1 + \theta_{23}z_2 + \theta_{33}z_3 \equiv_p 242. \end{aligned}$$

When evaluated at z, the polynomials W_k mod p computed by Algorithm 1 yield the same values w_k above. But here we obtained the w_k without computing the W_k, using just $O(h)$ operations to compute them directly from the y_i and the θ_{ik}. Most importantly, the CRT data used to compute the w_k mod q only consumes $O(m \log q)$ space, versus $O(h \log q)$ for the W_k mod q.

After computing the w_k mod p for all $p \in S$, the explicit CRT (mod q) yields:

$$w_0 \equiv_q 180694, \quad w_1 \equiv_q 270105, \quad w_2 \equiv_q 92440, \quad w_3 \equiv_q 110998.$$

We then evaluate $V'(y)$ mod q and use the w_k to compute

$$\begin{aligned} U(X, y) &= X^5 + yX^4 + \frac{1}{V'(y)}(w_3X^3 + w_2X^2 + w_1X + w_0) \\ &\equiv_q X^5 + 336976X^4 + 556976X^3 + 849678X^2 + 363260X + 95575, \end{aligned}$$

and find that $x = 590272$ is a root of $U(X, y)$, and hence of H_D, modulo q.

4.2. *Time complexity*

To simplify our analysis we shall initially assume that

$$m \log q = O(|D|^{1/2} \log |D|). \tag{15}$$

Depending on $m = h/|G|$, this may allow $\log q$ to be exponentially larger than $\log |D|$, but here we have in mind the case where $\log q$ is polynomial in $\log |D|$; see § 4.4 for an approach better suited to large q. Assuming (15), our bound on $|S|$ is the same as in our analysis of Algorithm 1 in § 3.2. Under the GRH we have

$$|S| = O(|D|^{1/2} \log \log |D|), \tag{16}$$

which bounds the total expected number of iterations in steps 4 and 7.

As with Algorithm 1, the expected running time of Algorithm 2 is bounded by

$$O(|S|(T_{\text{find}} + T_{\text{enum}} + T_{\text{build}}) + T_{\text{root}}), \tag{17}$$

where T_{find}, T_{enum}, T_{build}, and T_{root} are as defined in § 3.2. The term T_{find} is the same for both algorithms, and the term T_{enum} is doubled in Algorithm 2. The bound $O(h \log^2 h \log^{1+\epsilon} \log p)$ on T_{build} in Algorithm 1 becomes

$$T_{\text{build}} = O(m \log^2 m \log^{1+\epsilon} p + h \log^2 n \log^{1+\epsilon} p) \tag{18}$$

for Algorithm 2, with $p = \max S$. The first term in (18) is the time to build V, while the second is the time to compute the θ_{ik} and w_k. For suitable n, say $n = \log^c h$ for some $c > 2$, Algorithm 2 effectively reduces T_{build} by a factor of $\log^2 h$. There is also a minor improvement in T_{root}; the bound given in (14) becomes

$$T_{\text{root}} = O((m + n) \log h \log^{2+\epsilon} q). \tag{19}$$

https://doi.org/10.1112/S1461157012001015 Published online by Cambridge University Press

In both the GRH-based and heuristic complexity analyses of [42, § 7], the bound for T_{enum} dominates the sum $T_{\text{find}} + T_{\text{enum}} + T_{\text{build}}$. Thus a better bound on T_{build} does not improve the worst-case complexity. However, the worst-case scenario is atypical, and for almost all D (a set of density 1) this sum is dominated by T_{build}. This is heuristically argued in [42, § 7], and it can be proven using [6] and assuming the GRH (but we will not do so here).

To remove the worst-case impact of T_{enum} (and also T_{find}), let us consider the performance of Algorithm 2 on a suitably restricted set of discriminants. Fix positive constants $c_1, \dots, c_7$, and ϵ. For a positive real parameter α, let $\mathcal{D}(\alpha)$ denote the set of negative discriminants D with the following properties.

(i) The set of integers $p \leqslant c_1|D| \log^{1+\epsilon} |D|$ satisfying $4p = t^2 - v^2 D$ with $v \geqslant c_2 \log^{1/2+\epsilon/3}|D|$ contains at least $c_3|D|^{1/2} \log^{\epsilon/2} |D|$ primes.

(ii) There are at least two primes $\ell \leqslant c_4 \log^{1/2} |D|$ for which $(\frac{D}{\ell}) = 1$.

(iii) There is a divisor of $h = h(D)$ in the interval $[c_5 \log^{\alpha} h, \exp(c_6 \log^{3/4} h)]$.

Conditions (i) and (ii) ensure that for $D \in \mathcal{D}(\alpha)$, both T_{find} and T_{enum} are bounded by $O(|D|^{1/2} \log^{5/2+\epsilon} |D|)$; we refer to [42, § 7] for details. Provided that $\alpha > 1/2$, for $D \in \mathcal{D}(\alpha)$ we can choose G so that

$$c_5 \log^{1/2} h \leqslant m \leqslant \exp(c_6 \log^{3/4} h) \quad \text{and} \quad n \leqslant h/(c_5 \log^{1/2} h),$$

where $n = |G|$ and $mn = h$. Applying (18), we see that T_{build} is then also bounded by $O(|D|^{1/2} \log^{5/2+\epsilon} |D|)$.

To ensure that our assumption in (15) is satisfied, let us define

$$\mathcal{P}_D(\alpha) = \{q \in \mathcal{P}_D : \log q \leqslant c_7 \log^{1+\alpha} |D|\}.$$

This definition is more restrictive than necessary, but for simplicity we impose a uniform bound. For $q \in \mathcal{P}_D(\alpha)$ we then have $T_{\text{root}} = O(|D|^{1/2+\epsilon})$, which is a negligible component of (17). This yields the following proposition.

PROPOSITION 1. *Assume the GRH. For all $\alpha > 1/2$, $D \in \mathcal{D}(\alpha)$, and $q \in \mathcal{P}_D(\alpha)$ there is a choice of G for which the expected running time of Algorithm 2 is $O(|D| \log^{5/2+\epsilon} |D|)$.*

PROPOSITION 2. *For all $\alpha < \log 2$, the set $\mathcal{D}(\alpha)$ has density 1 in the set of all imaginary quadratic discriminants.*

Proof. It suffices to show that each of the properties (i), (ii), and (iii) hold for a set of discriminants D with density 1 (in the set of all imaginary quadratic discriminants). For (i), this follows from [6, Theorem 2].

For (ii), consider just the odd primes $\ell_1, \dots, \ell_k$ less than $\log \log |D| \leqslant c_4 \log^{1/2} |D|$, for sufficiently large $|D|$. The proportion of congruence classes modulo $L = \prod \ell_i$ corresponding to integers x for which $(\frac{x}{\ell_i}) \neq 1$ for all but at most one ℓ_i is equal to

$$\prod \frac{\ell_i + 1}{2\ell_i} + \sum_i \frac{\ell_i - 1}{2\ell} \prod_{j \neq i} \frac{\ell_j + 1}{2\ell_j} < (2/3)^k + (k/2)(2/3)^{k-1} = o(1).$$

Thus the number of discriminants in the interval $[-2D, -D]$ that do not satisfy property (ii) is $o(|D|)$.

For (iii), recall that for any $\epsilon > 0$ and almost all integers n there are at least $(1 - \epsilon) \log \log n$ distinct prime divisors of n; see [29, Theorem 431]. Therefore almost all discriminants D have at least $k = (1 - \epsilon) \log \log |D| - 1$ distinct odd prime factors, and for all such discriminants, $h = h(D)$ is divisible by 2^{k-1}; see [20, Lemma 5.6.8]. As shown by Siegel, $\log h = (1/2 + o(1)) \log |D|$, thus for all $\alpha < \log 2$ we can choose $\epsilon > 0$ so that $2^{k-1} > c_5 \log^{\alpha} h$ for all sufficiently large $|D|$. □

https://doi.org/10.1112/S1461157012001015 Published online by Cambridge University Press

Propositions 1 and 2 together imply that, under the GRH, for almost all discriminants $D < 0$ one can find a root of $H_D \bmod q$ in $O(|D| \log^{5/2+\epsilon} |D|)$ expected time, for all $q \in \mathcal{P}_D(\alpha)$ with $1/2 < \alpha < \log 2$.

We note that the time required to identify and select a suitable $G \subset \mathrm{cl}(\mathcal{O})$ is negligible by comparison. When D is fundamental we can obtain a set of generators for the class group $\mathrm{cl}(\mathcal{O})$ using ideal class representatives of prime norm bounded by $6 \log^2 |D|$, under the GRH [**5**], and for non-fundamental D we also include ideals of prime-power norm for primes dividing the conductor, of which there are $O(\log |D|)$. Given a generating set S for $\mathrm{cl}(\mathcal{O})$ of size $O(\log^2 |D|)$, we can apply generic algorithms to compute the group structure and an explicit basis for $\mathrm{cl}(\mathcal{O})$ consisting of elements of prime-power order, in time $O(h^{1/2+\epsilon})$, where $h = |\mathrm{cl}(\mathcal{O})| = h(D)$, via [**43**, Proposition 4][†]. Once this has been done, it is easy to construct an explicit basis for a subgroup G of any desired order n dividing h.

Remark 3. It is usually better to choose G to optimize the height bound B rather than making the choice required by Proposition 1. Heuristically, this yields a better improvement in the time complexity, a factor of nearly $\log |D|$ rather than $\log^{1/2} |D|$, see Heuristic Claim 1 in § 5.

4.3. *Space complexity*

For convenience we assume the GRH and the restriction (15) on the size of q. An alternative approach with a weaker restriction on q is given in the next section. Under these assumptions, $\log p \sim \log |D|$ for all $p \in S$, and $|S| = O(|D|^{1/2} \log\log |D|)$, as in (16). A straightforward implementation of Algorithm 2 yields a space complexity of

$$O(|S| \log |D| + |S| \log q + h \log |D| + (m+n) \log q). \tag{20}$$

The first term of (20) represents storage for the set S, the second term is storage for precomputed data used to apply the explicit CRT (mod q), the third term is storage for $\mathrm{Ell}_{\mathcal{O}}(\mathbf{F}_p)$, and the last term is storage for the n values w_k and the m coefficients of V that are computed via the explicit CRT (mod q). The second term dominates, and we can immediately bound (20) by $O(|D|^{1/2} \log\log |D| \log q)$, which is the same as the space complexity of Algorithm 1.

However, the only essential term in (20) is the last one, which may be as small as $O(|D|^{1/4} \log q)$. We now show how to eliminate the first three terms in (20). In practice the most useful term to eliminate (or reduce) is the second one, which requires only a very minor change, but we consider each in turn.

4.3.1. *The first term.* We wish to avoid storing the entire set S at any one time. Our strategy is to process S in batches of size $O(|D|^c)$, for some positive $c < 1/4$.

In [**42**] the set S is chosen by enumerating a larger subset $S_z \subset \mathcal{P}_D$ defined by a parameter z that depends on the bound B. The primes in S_z satisfy the norm equation $4p = t^2 - v^2 D$, where v is $O(\log^{3+\epsilon} |D|)$, and the bound on t depends on v, $h(D)$, and z, but is in any case $O(|D|^{1/2+\epsilon})$. As an alternative to the sieving approach of [**42**, Algorithm 2.1], we enumerate S_z by running through all the integers of the form $(t^2 - v^2 D)/4$, with v and t suitably bounded, and applying a polynomial-time primality test [**2**] to each. This takes $O(|D|^{1/2+\epsilon})$ time and negligible space.

Each prime p in S_z is assigned a 'rating' $r(p)$, that reflects the expected cost of finding $j_1 \in \mathrm{Ell}_{\mathcal{O}}(\mathbf{F}_p)$. The details are not important here, but we may assume that the positive real numbers $r(p)$ are distinctly represented using a precision of $O(\log |D|)$ bits. Let $r_{\max}$ be the

[†]Note that the exponent $\mathrm{lcm}_{\alpha \in S} |\alpha|$ of $\mathrm{cl}(\mathcal{O})$ can be computed in time $O(h^{1/2+\epsilon})$; see [**41**].

https://doi.org/10.1112/S1461157012001015 Published online by Cambridge University Press

largest (worst) rating among the primes in S_z. For a suitable $r \in [0, r_{\max}]$, we then let S consist of the primes in S_z with ratings bounded by r, where r is chosen so that S has the appropriate size. We can determine such an r space-efficiently using a binary search on the interval $[0, r_{\max}]$, enumerating S_z at most $\lceil \log_2 |S_z| \rceil$ times. Similarly, we can partition S into batches of size $|D|^c$ by partitioning the interval $[0, r]$ into subintervals whose endpoints are determined using a binary search. The total cost of computing this partition is $O(|D|^{1-c+\epsilon})$, and this also bounds the cost of enumerating all the batches in S. Thus the time complexity is negligible and we use $o(|D|^{1/4})$ space.

4.3.2. *The second term.* Let M be the product of the primes p_i in S. As described in § 2.3, when applying the explicit CRT (mod q), we precompute integers $M_i = M/p_i \bmod q$ and $a_i \equiv (M/p_i)^{-1} \bmod p_i$; this would normally be done in step 3 of Algorithm 2. The total size of the M_i is $O(|S| \log q)$, which accounts for the second term of (20). Rather than computing the M_i in step 3, we just compute $M \bmod q$ in step 3, and when we need M_i during a CRT update step for p_i (steps 4d and 7d), we invert $p_i \bmod q$ and multiply to obtain $M_i = (M/p_i) \bmod q$. This reduces the second term of (20) from $O(|S| \log q)$ to $O(|S| \log |D| + \log q)$, which is the space used by the a_i and $M \bmod q$.

With the changes described in § 4.3.1, we compute the primes p_i in S in batches of size $|D|^c$, where $c < 1/4$. We now do the same with the a_i, computing the a_i for each batch of primes p_i as follows. If N is the product of the primes in the batch, then it suffices to compute the integer (M/N) modulo N, and then simultaneously compute the product of (M/N) and (N/p_i) modulo p_i, for all the primes p_i in the batch, to obtain the a_i. The a_i (and the p_i) for a given batch are stored only as long as it takes to process the batch; this means that they are computed twice in Algorithm 2: once in step 4 and then again in step 7. Using a product tree for each batch, it takes $O(|D|^{1-c+\epsilon})$ time to compute the a_i for all the batches, which is negligible. With this change, the second term of (20) is reduced to $o(|D|^{1/4}) + O(\log q)$.

The only data that is retained once the processing of a given batch of primes has been completed are two values associated to each coefficient c that is to be computed modulo q (the m coefficients of $V \bmod q$ in step 5, and the n values $w_k \bmod q$). These two values are the partial sums $\sum c_i a_i M_i$ and $\sum c_i a_i / p_i$, where the first is computed modulo q and the latter is stored with a precision of $O(\log q)$ bits, as described in § 2.3. These two values are updated as each prime p_i is processed (across all the batches), and never require more than $O(\log q)$ bits. There are a total of $m + n$ coefficients, which accounts for the fourth term $O((m + n) \log q)$ listed in (20), which also bounds the space required to perform the CRT postcomputation; using the accumulated partial sums, the postcomputation is effectively just a single subtraction to evaluate (3).

4.3.3. *The third term.* Steps 4b and 7b of Algorithm 2 both enumerate the G-orbits of $j_1 \in \mathrm{Ell}_{\mathcal{O}}(\mathbf{F}_p)$, using the polycyclic presentation Γ. Let j_{ik} denote the kth element of the G-orbit $G_i = (j_{i1}, \ldots, j_{in})$, where i ranges from 1 to m and k ranges from 1 to n. There are a total of $h = mn$ values j_{ik}, and these account for the $O(h \log |D|)$ third term in (20), using the GRH bound $\log p = O(\log |D|)$. In order to reduce the space required, we need to process the j_{ik} as they are enumerated, rather than storing them all. Our basic strategy is to order the enumeration so that we compute the G-orbits one by one and discard each G-orbit once it has been processed (in fact, we need to enumerate the G-orbits twice in step 7 in order to achieve this). Depending on the presentation Γ, enumerating the G-orbits efficiently may present some complications. These will be addressed below. We first consider how to process the G-orbits in a space-efficient manner as they are enumerated.

Let us recall the values we must derive from the j_{ik}. In each iteration of step 4 we compute m values $y_i \bmod p$, each of which is a linear combination of elementary symmetric functions applied to $G_i = (j_{i1}, \ldots, j_{in})$. We then compute the polynomial $V \bmod p$ whose roots are

https://doi.org/10.1112/S1461157012001015 Published online by Cambridge University Press

the y_i. In each iteration of step 7 we again compute the y_i mod p, and also the coefficients θ_{ik} of the polynomials

$$P_i(X) = \prod_{k=1}^{n} = (X - j_{ik}) = \sum_{k=0}^{n} \theta_{ik} X^k$$

defined in (5). Up to a sign, the θ_{ik} are just the elementary symmetric functions of $j_{i1}, \ldots, j_{in}$, so we can derive each y_i from the θ_{ik}. The y_i are then used to compute V mod p (again), and also the values $z_i = V(z)/(z - y_1)$, where $z = \varphi(y)$ mod p is the reduction of the integer $\varphi(y)$ corresponding to the root y of V mod q computed in step 6. Finally, we compute the values $w_k = \sum_{i=1}^{m} \theta_{ik} z_i$, as described in § 4.1.

The space required by the y_i, the z_i, the w_k and the polynomials V and V' is just $O((m+n)\log p)$, which is within our desired complexity bound. We now explain how to process the G-orbits in a way that achieves this space complexity. For each G_i we compute the polynomial P_i with coefficients θ_{ik}, use the θ_{ik} to compute y_i, and then discard G_i and the θ_{ik}. In step 4 this is all that is required; once all the G-orbits have been processed we compute $V(Y) = \prod_{i=1}^{m}(Y - y_i)$. In step 7 we proceed as in step 4, and after computing V and the y_i we compute V' and the values $z_i = V(z)/(z - y_i)$. We then enumerate the G-orbits a second time, recompute the θ_{ik} as above, and then update each of n partial sums $w_k = \theta_{ik} z_i$ by adding the term $\theta_{ik} z_i$. The space required to process a G-orbit is $O(n \log p)$ for the j_{ik} and the θ_{ik}, which are then discarded, plus a total of $O((m+n)\log p)$ space used to store the value y_i and z_i, and the partial sums w_k that are retained. Thus we can process all the G-orbits using $O((m+n)\log p)$ space.

We now consider the enumeration of the G-orbits. If the polycyclic presentation Γ for $\mathrm{cl}(\mathcal{O})$ uses the sequence of ideal classes $[\mathfrak{l}_1], \ldots, [\mathfrak{l}_r]$ and the chosen subgroup G is generated by a prefix of Γ, say

$$G = \langle [\mathfrak{l}_1], \ldots, [\mathfrak{l}_d] \rangle, \tag{21}$$

for some $d < r$, then the elements $j_{i1}, \ldots, j_{in}$ of the G-orbit G_i will appear consecutively in the enumeration of $\mathrm{Ell}_{\mathcal{O}}(\mathbf{F}_p)$ obtained using Γ and no special processing is required. But the situation in (21) is very special, much more so than condition (9) given in § 2.6. Indeed, it forces G to be trivial if $r = 1$. We can ensure that G has the form in (21) if we construct Γ by computing a polycyclic presentation for G and extending it to a polycyclic presentation for $\mathrm{cl}(\mathcal{O})$. Unfortunately, the norms arising in a polycyclic presentation for a proper subgroup of $\mathrm{cl}(\mathcal{O})$ may be very large: $\Omega(|D|^{1/3})$ in the counterexample of [**42**, § 5.3].

To address this, we compute the action of ideals with uncomfortably large norm by representing them as a product of ideals with small norms. Assuming the GRH, it follows from [**16**, Theorem 2.1] that every element $[\mathfrak{a}]$ of $\mathrm{cl}(\mathcal{O})$ can be expressed in the form $[\mathfrak{a}] = [\mathfrak{p}_1 \ldots \mathfrak{p}_t]$, where the $\mathfrak{p}_i$ are ideals of prime norm bounded by $\log^c |D|$, for any $c > 2$, and $t = \lceil C \log h / \log\log |D| \rceil = O(\log |D|)$, for some constant C that depends on c. We can find such a representation in $O(h^{1+\epsilon})$ expected time (using negligible space), by simply generating random products of the form $[\mathfrak{p}_1 \ldots \mathfrak{p}_t]$ until we find $[\mathfrak{a}]$[†]. After precomputing such a representation in step 2, we can compute the action of any element of $\mathrm{cl}(\mathcal{O})$ on any element of $\mathrm{Ell}_{\mathcal{O}}(\mathbf{F}_p)$ in $O(\log^{6+\epsilon} |D|)$ expected time, allowing us to efficiently handle arbitrarily large norms in Γ.

In contrast to the first two terms, removing the third term from (20) may increase the overall running time significantly. The time complexity is increased by a logarithmic factor, but in return, the space complexity may be reduced by an exponential factor. We did not make this trade-off in our implementation, as the space complexity of $O(h \log h + (m+n)\log q)$

[†]This can be improved to $O(h^{1/2+\epsilon})$ time and $O(h^{1/2-\epsilon})$ space using a birthday-paradox approach with a time/space trade-off, but we don't need to do this.

https://doi.org/10.1112/S1461157012001015 Published online by Cambridge University Press

achieved by simply optimizing the second term is already more than sufficient for the range of D used in the examples of § 6. However for larger computations, increasing the running time by a polylogarithmic factor in order to reduce the space by an exponential factor may be very attractive, especially in a parallel implementation.

Eliminating the first three terms of (20) leads to the following proposition, which was summarized in the introduction. The constraints on m and q ensure that (15) is satisfied.

PROPOSITION 3. *Assume the GRH, and fix $\delta \geqslant 0$. Let D be a discriminant with class number $h = mn$ and let $q \in \mathcal{P}_D$, such that $m \leqslant O(|D|^{1/2-\delta})$ and $\log q = O(|D|^{\delta} \log |D|)$. With modifications 4.3.1-3, the expected running time of Algorithm 2 on inputs D and q is $O(|D| \log^{6+\epsilon} |D|)$, using $O\big((m+n)\log q\big)$ space.*

4.4. *Handling large q*

When q is not constrained by (15), the bound $mq^{m-1}B$ used by Algorithm 2 to determine the size of S may lead to a significant increase in the running time of Algorithm 2 relative to Algorithm 1, which just uses the bound B. To better handle large q in a space-efficient manner, we make a minor modification to Algorithm 2 that allows us to use the bound mqB instead. Unless q is extraordinarily large, this will not be significantly different than using the bound B. We are forced to give up the improved bound on T_{build} in (18), but the resulting algorithm will still have a running time that is at worst twice that of Algorithm 1, and will typically be only slightly slower.

The key is to avoid exponentiating $\varphi(y)$ in $\mathbf{F}_p$. Instead we compute all of the powers $y, y^2, y^3, \ldots, y^{m-1}$ in $\mathbf{F}_q$, and then lift these to integers $\varphi(y), \ldots, \varphi(y^{m-1})$ in the interval $[0, q-1]$, which can be reduced modulo p. This is done between steps 6 and 7 of Algorithm 2. We now modify step 5c to compute the coefficients of $W_k = \sum a_{ik} Y^i \bmod p$ as in Algorithm 1, and then compute values $w'_k \bmod p$ via

$$w'_k = a_{m-1,k}\varphi(y^{m-1}) + a_{m-2,k}\varphi(y^{m-2}) + \ldots + a_{1,k}\varphi(y) + a_{0,k},$$

that we use instead of $w_k = W_k(\varphi(y)) \bmod p$. Note that the integers w'_k are not equal to the integers $W_k(\varphi(y))$, but we have $w'_k \equiv W_k(\varphi(y)) \bmod q$, which is all that is required, even though w_k and w'_k will typically be distinct modulo p.

We may combine this approach with any of the space optimizations considered in the previous section. When q is very large, the third term of (20) is likely to be dominated by the others, so we only optimize the first two terms of (20). Provided $\log q = O(|D|^{1/2})$, the analysis in [**42**, § 7] then yields an upper bound on the running time of this modified version of Algorithm 2.

PROPOSITION 4. *Assume the GRH. Let D be a discriminant with class number $h = mn$, and let $q \in \mathcal{P}_D$ satisfy $\log q = O(|D|^{1/2})$. With modifications 4.3.1-2 and 4.4, the expected running time of Algorithm 2 on inputs D and q is $O\big(|D| \log^{5+\epsilon} |D|\big)$, using $O\big((m+n)\log q + h \log h\big)$ space.*

5. *Height bounds*

In this section we derive an upper bound B on the absolute values of the integer coefficients of V and W_k defined in § 2.5. More precisely, we compute a height bound b on the maximum bit-length of any coefficient occurring in V or W_k. This is used by Algorithms 1 and 2 to choose a set of CRT primes S whose product exceeds $4B = 2^{b+2}$.

We first derive a general height bound $b_{\max}$ that depends only on D and can be used with any choice of the subgroup $G \subset \mathrm{cl}(\mathcal{O})$ and the random symmetric function

https://doi.org/10.1112/S1461157012001015 Published online by Cambridge University Press

$s = e_1 + c_2 e_2 + \ldots + c_n e_n$ constructed in Algorithms 1 and 2. Under the GRH, we have $b_{\max} = O(|D|^{1/2} \log |D| \log\log |D|)$, which is all that is needed for the proofs of Propositions 1–4.

We then fix $s = e_1$ and derive height bounds that depend not only on D, but also on the subgroup G. As may be seen in Table 2, the actual heights can vary significantly with G. An optimal choice of G may improve the performance of both Algorithms 1 and 2 substantially, provided that we have a height bound that accurately reflects the impact of G. Heuristically, we expect to reduce b by a factor of $\log h / \log\log h$, on average, by computing a customized b for each candidate subgroup G and choosing the best one. Of course the optimal choice of G depends not only on b, but also on how the size of G impacts the complexity of building polynomials and finding roots, but when $\log q \ll h$ the height bound is usually the most critical factor.

Throughout this section we work with j-invariants, which allows us to use rigorous (and quite accurate) bounds on their size. Heuristic bounds for other class invariants can be obtained by scaling linearly, as discussed in § 6. We note that all the bounds we derived in this section hold unconditionally; the GRH and the heuristic analysis in § 5.4 are only used to obtain asymptotic growth estimates.

5.1. *Height bound derivations*

As in § 2.5, let $G = \{\beta_1, \ldots, \beta_n\}$ be a subgroup of $\mathrm{cl}(\mathcal{O})$ with cosets $\alpha_1 G, \ldots, \alpha_m G$, so that every element of $\mathrm{cl}(\mathcal{O})$ is of the form $\alpha_i \beta_k$ with $1 \leqslant i \leqslant m$ and $1 \leqslant k \leqslant n$. We may uniquely represent $\alpha_i \beta_k$ by a primitive reduced binary quadratic form $A_{ik}x^2 + B_{ik}xy + C_{ik}y^2$ with discriminant D. If τ_{ik} denotes the complex number $(-B_{ik} + \sqrt{D})/(2A_{ik})$, we have

$$H_D(X) = \prod_{i,k} (X - j(\tau_{ik})),$$

where $j(z)$ is the classical modular function. We assume without loss of generality that α_1 and β_1 are the identity element, and fix $x = j(\tau_{11})$ so that $[\alpha_i \beta_k]x = j(\tau_{ik})$. We shall use the explicit bound

$$|j(\tau_{ik})| \leqslant \exp(\pi \sqrt{|D|}/A_{ik}) + 2114.567 \tag{22}$$

proved in [**21**, p. 1094], and define b_{ik} to be the logarithm in base 2 (denoted 'lg') of the RHS of (22), and we note that $b_{ik} > 0$. We define the *height* $\mathrm{ht}(F)$ of a nonzero polynomial $F(X) = \sum c_j X^j$ in $\mathbf{C}[X]$ by $\mathrm{ht}(F) = \lg \max |c_j|$. We seek an upper bound b on $\max\{\mathrm{ht}(V), \mathrm{ht}(W_k)\}$ in terms of the b_{ik}.

TABLE 2. *Actual heights for various $G \subset \mathrm{cl}(\mathcal{O})$ with $D = -221606831$.*

$\lvert G\rvert$	Bits	$\lvert G\rvert$	Bits	$\lvert G\rvert$	Bits	$\lvert G\rvert$	Bits
1	1983568	35	1017514	182	639986	910	395909
2	1737305	39	955880	195	672404	1001	444642
3	1600984	42	959237	210	649274	1155	413905
5	1464042	55	879633	231	607751	1365	392521
6	1430692	65	841574	273	603539	1430	402990
7	1354754	66	780769	286	540873	2002	414360
10	1286551	70	877290	330	531985	2145	422627
11	1235548	77	791884	385	522887	2310	401968
13	1202022	78	760840	390	540120	2730	409766
14	1188816	91	756960	429	525472	3003	436780
15	1195102	105	773983	455	430383	4290	471475
21	1093207	110	677448	462	487746	5005	507403
22	962794	130	720919	546	492453	6006	549648
26	1010539	143	697728	715	452019	10010	756598
30	1006310	154	616795	770	429293	15015	1039684
33	998157	165	678832	858	437618	30030	1983568

https://doi.org/10.1112/S1461157012001015 Published online by Cambridge University Press

The largest b_{ik} is b_{11}, since $\alpha_1\beta_1$ is the identity and we therefore have $A_{11} = 1$. The bound (22) is nearly tight (one can prove a similar lower bound), and this implies that $b_{11} \approx \lg(e)\pi\sqrt{|D|}$. This is about ten times the typical value of h, so we shall not be concerned with optimizing terms that are asymptotically smaller than h, which we may assume includes both m and n, and even $m \lg n$ and $n \lg m$.

As in § 2.5, the θ_{ik} are defined via

$$P_i(X) = \prod_{k=1}^{n} \big(X - j(\tau_{ik})\big) = \sum_{k=1}^{n} \theta_{ik} X^k, \tag{23}$$

where $\prod_{i=1}^{m} P_i = H_D$. This yields the bound

$$\mathrm{ht}(P_i) \leqslant n + \sum_{k=1}^{n} b_{ik}. \tag{24}$$

For $V(Y) = \prod_{i=1}^{m}(Y - y_i)$, with $y_i = s(j(\tau_{i1}), \ldots, j(\tau_{in}))$, we assume that s is of the form $s = \pm(e_1 + c_2 e_2 + \ldots + c_n e_n)$ with $c_k \geqslant 0$, and define $z_i = |s(2^{b_{i1}}, \ldots, 2^{b_{in}})|$. We then have $|y_i| \leqslant z_i$, with $z_i > 1$. Thus

$$\mathrm{ht}(V) \leqslant m + \sum_{i=1}^{m} \lg z_i. \tag{25}$$

For the nonzero polynomials $W_k(Y) = \sum_{i=1}^{m} \theta_{ik} \prod_{\hat{\imath}\neq i}(Y - y_{\hat{\imath}})$, we note that

$$\sum_{i=1}^{m} |\theta_{ik}| 2^m \prod_{\hat{\imath}\neq i} z_{\hat{\imath}}$$

is positive, and bounds every coefficient of W_k. We have

$$\begin{aligned} \mathrm{ht}(W_k) &\leqslant \lg\left(\sum_{i=1}^{m} |\theta_{ik}| 2^m \prod_{\hat{\imath}\neq i} z_{\hat{\imath}}\right) \leqslant \lg\left(m \max_i |\theta_{ik}| 2^m \prod_{\hat{\imath}\neq i} z_{\hat{\imath}}\right) \\ &\leqslant m + \lg m + \max_i \left(\mathrm{ht}(P_i) + \sum_{\hat{\imath}\neq i} \lg z_{\hat{\imath}}\right). \end{aligned} \tag{26}$$

This expression does not depend on k, thus it applies to every nonzero W_k.

5.1.1. *A general height bound.* For any choice of $s = e_1 + c_2 e_2 + \ldots + c_n e_n$ with $0 \leqslant c_k < 2m^2$, we have $z_i \leqslant 2m^2 n 2^{n-1} 2^{b_{i1}+\ldots+b_{in}}$, using $\binom{n}{k} \leqslant 2^{n-1}$, and therefore

$$\lg z_i \leqslant n + \lg n + 2\lg m + \sum_{k=1}^{n} b_{ik}. \tag{27}$$

From (25) we obtain

$$\mathrm{ht}(V) \leqslant m + mn + m\lg n + 2m\lg m + \sum_{i,k} b_{ik} \leqslant 3h + 2h\lg h + \sum_{i,k} b_{ik}, \tag{28}$$

where we have used $h = mn \geqslant m \lg n$. Using the crude bounds $\max_i \sum_k b_{ik} \leqslant \sum_{i,k} b_{ik}$ and $\sum_{\hat{\imath}\neq i} \lg z_{\hat{\imath}} \leqslant \sum_i \lg z_i$, and applying (24) and (27) to (26), we obtain

$$\begin{aligned} \mathrm{ht}(W_k) &\leqslant m + \lg m + n + mn + m\lg n + 2m\lg m + 2\sum_{i,k} b_{ik} \\ &\leqslant 5h + 2h\lg h + 2\sum_{i,k} b_{ik}. \end{aligned} \tag{29}$$

https://doi.org/10.1112/S1461157012001015 Published online by Cambridge University Press

Since the bound in (29) dominates the bound in (28), we define

$$b_{\max} = 5h + 2h \lg h + 2 \sum_{i,k} b_{ik}, \tag{30}$$

where i runs from 1 to m and k runs from 1 to n.

Recall that, under the GRH, we have the bounds $h = O(|D|^{1/2} \log\log |D|)$ and $\sum_{i,k} 1/A_{ik} = O(\log |D| \log\log |D|)$, as noted in § 2.4. These imply

$$b_{\max} = O(|D|^{1/2} \log |D| \log\log |D|). \tag{31}$$

5.1.2. *Optimized height bounds.* We now fix $s = e_1$, which implies $z_i = \sum_{k=1}^{n} b_{ik}$. This choice of s minimizes our height bound and simplifies the calculations.

We have

$$\lg z_i \leqslant \lg\left(\sum_{k=1}^{n} 2^{b_{ik}}\right) \leqslant \lg n + \max_k b_{ik}, \tag{32}$$

and from (25) we find that

$$\mathrm{ht}(V) \leqslant m + m \lg n + \sum_{i=1}^{m} \max_k b_{ik}. \tag{33}$$

Applying (24) and (32) to (26) yields

$$\begin{aligned}
\mathrm{ht}(W_k) &\leqslant \lg m + m + \max_i \left(n + \sum_k b_{ik} + \sum_{\hat{\imath} \neq i} (\lg n + \max_k b_{\hat{\imath}k}) \right) \\
&\leqslant \lg m + m + n + m \lg n + \max_i \left(\sum_k b_{ik} + \sum_{\hat{\imath}} \max_k b_{\hat{\imath}k} - \max_k b_{ik} \right) \\
&\leqslant \lg m + m + n + m \lg n + \sum_i \max_k b_{ik} + \max_i \left(\sum_k b_{ik} - \max_k b_{ik} \right),
\end{aligned}$$

where i runs from 1 to m and k runs from 1 to n. This bound dominates the bound in (33), so it bounds $\mathrm{ht}(V)$ as well as $\mathrm{ht}(W_k)$. Thus we define

$$b = \lg m + m + n + m \lg n + \sum_i \max_k b_{ik} + \max_i \left(\sum_k b_{ik} - \max_k b_{ik} \right) \tag{34}$$

as our height bound for G, which we typically round up to the nearest integer.

5.2. *Example*

Returning to our example with $D = -971$ and $h(D) = 15$, let us compute b for the subgroup $G \subset \mathrm{cl}(\mathcal{O})$ of order $n = 5$. We can use the same polycyclic presentation $[\mathfrak{l}_1], [\mathfrak{l}_2]$ for $\mathrm{cl}(\mathcal{O})$ as before, where the ideals $\mathfrak{l}_1$ and $\mathfrak{l}_2$ have norms $\ell_1 = 3$ and $\ell_2 = 5$ and $[\mathfrak{l}_1]$ generates G, but now we compute directly in the class group, using composition of binary quadratic forms [14] rather than computing isogenies. In the notation of § 5.1, we have $\beta_k = [\mathfrak{l}_1^{k-1}]$ and $\alpha_i = [\mathfrak{l}_2^{i-1}]$. Enumerating $\mathrm{cl}(\mathcal{O})$ yields the approximate values

$$\begin{array}{llllll}
b_{11} = 141.23, & b_{12} = 47.08, & b_{13} = 15.75, & b_{14} = 15.75, & b_{15} = 47.08, \\
b_{21} = 28.25, & b_{22} = 11.45, & b_{23} = 20.18, & b_{24} = 11.96, & b_{25} = 11.45, \\
b_{31} = 11.45, & b_{32} = 11.96, & b_{33} = 20.18, & b_{34} = 11.45, & b_{35} = 28.25,
\end{array}$$

where each row corresponds to a coset of G. The value of b_{22}, for example, is computed using $A_{22} = 15$, since $\alpha_2 \beta_2 = [\mathfrak{l}_2 \mathfrak{l}_1]$ is represented by the reduced binary quadratic form $15X^2 + 13XY + 19Y^2$, and we have

$$b_{22} = \lg(\exp(\pi\sqrt{971}/15) + 2114.567) \approx 11.45.$$

https://doi.org/10.1112/S1461157012001015 Published online by Cambridge University Press

To compute b we just need the sum s_i and maximum t_i of b_{ik} over k. These can be computed during the enumeration and stored using $O(m \log |D|)$ space. We have $s_1 = 266.89$, $s_2 = 83.28$, $s_3 = 83.38$, and $t_1 = 141.23$, $t_2 = 28.25$, $t_3 = 28.25$. The t_i sum to 197.73, the maximum of $s_i - t_i$ is 125.65. Applying (34) yields

$$b \approx \lg 3 + 5 + 3 + 3 \lg 5 + 197.73 + 125.65 \leqslant 340,$$

and our height bound is 340 bits.

As noted earlier, we may compute V and the W_k over $\mathbf{Z}$ using Algorithm 1, by computing them modulo some $q \geqslant 2^{b+1}$. We find that $\max\{\mathrm{ht}(V), \mathrm{ht}(W_k)\}$ is in fact about 324, within five percent of our computed bound b.

For comparison, if we instead choose G to be the subgroup of order 3, we obtain $s_1 = 165.15$, $s_2 = 55.45$, $s_3 = 78.71$, $s_4 = 78.71$, $s_5 = 55.45$, and $t_1 = 141.23$, $t_2 = 28.25$, $t_3 = 47.08$, $t_4 = 47.08$, $t_5 = 28.25$. The t_i sum to 291.89, the maximum of $s_i - t_i$ is 31.63, and (34) becomes

$$b \approx \lg 5 + 3 + 5 + 5 \lg 3 + 291.89 + 31.63 \leqslant 342.$$

If we make G trivial, we have $t_i = s_i = b_{i1}$, the sum of the t_i is 421.51, the maximum of $s_i - t_i$ is zero, and we get $b = 438$, which is nearly the same as the value 434 one obtains from [**42**, Lemma 8], which uses a more careful analysis than we do here.

5.3. *Computational complexity of optimizing the height bound*

We can compute a basis $(\gamma_1, \ldots, \gamma_r)$ for the class group $\mathrm{cl}(\mathcal{O})$ in $O(|D|^{1/4+\epsilon})$ time and space, under the GRH [**14**, Proposition 9.7.16]. We may assume that each γ_i has prime power order $\ell_i^{e_i}$. We then consider subgroups G of the form

$$G = \langle \gamma_1^{\ell_1^{d_1}} \rangle \times \ldots \times \langle \gamma_r^{\ell_r^{d_r}} \rangle, \tag{35}$$

with $0 \leqslant d_i \leqslant e_i$, which includes subgroups of every possible order n dividing $h = h(\mathcal{O})$. There are at most h such subgroups G, and enumerating the cosets of G takes $O(h \log^{1+\epsilon} h)$ time, using fast composition of forms [**38**].

Thus we can compute a height bound b for every subgroup G of the form in (35) in $O(h^2 \log^{1+\epsilon} h)$ time. This is $O(|D| \log^{1+\epsilon} |D|)$ under the GRH, which is dominated by our bounds for the running times of Algorithms 1 and 2. In fact, there are only $O(h^\epsilon)$ distinct orders that can arise among the candidate subgroups G, and if we restrict our attention to subgroups of the form in (9), there may be even fewer G to consider. In practice, the time spent optimizing b is completely negligible (and well worth the effort in any case).

As noted in the example, we only need $O(m \log |D|)$ space to compute the height bound for a given subgroup G, which is within the complexity bound of Proposition 2.

5.4. *Heuristic analysis*

Ignoring the minor terms in (34), the value

$$b^* = \sum_{i=1}^{m} \max_k b_{ik} + \max_i \left(\sum_{k=1}^{n} b_{ik} - \max_k b_{ik} \right)$$

closely approximates b. When $m = h$ or $n = h$ we have $b^* = \sum_{i,k} b_{ik}$, and in any case b^* is never greater than this sum. As with $b_{\max}$, this yields an asymptotic bound for b^* of $O(|D|^{1/2} \log |D| \log\log |D|)$, under the GRH, which then also bounds b. This is all that can be said in general, since h could be prime.

But h is rarely prime. This can occur only when $|D|$ is prime, and even then it is unlikely (by Cohen–Lenstra [**17**]). Let us consider the typical situation, where we are more or less free to

https://doi.org/10.1112/S1461157012001015 Published online by Cambridge University Press

choose the size of G, at least up to a constant factor[†]. Of course b^* depends on the particular choice of G, not just its order n, but to simplify matters we focus on n, and proceed to derive a heuristic estimate for b^* as a function of n and $m = h/n$.

Let us assume that the cosets G_i of G are ordered so that $\min_k A_{ik}$ is increasing with i (thus $G_1 = G$ contains the identity element $\alpha_1\beta_1$ with $A_{11} = 1$). As a heuristic, let us suppose that, on average, we have

$$\sum_{i,k} \frac{1}{A_{ik}} \approx \sum_{i,k} \frac{1}{i + (k-1)m} = \sum_{t=1}^{h} \frac{1}{t} \approx \log h.$$

That is, we view $\sum 1/A_{ik}$ as an approximation to a harmonic sum in which the terms corresponding to the ith coset of G appear at positions $i,\ i+m, \ldots, i+(n-1)m$. This heuristic is based on empirical data collected during the construction of Table 3, which involved analyzing the subgroups of more than 10 000 distinct class groups $\mathrm{cl}(\mathcal{O})$, with discriminants ranging from 10^5 to 10^{16}. We should emphasize that for any particular choice of $\mathrm{cl}(\mathcal{O})$ and G, the actual situation may deviate quite significantly from this idealized scenario, but if one averages over a large set of class groups and a large sample of their subgroups, one finds, for example, that the average rank of $\min_k A_{ik}$ among all the A_{ik} is approximately i, and we note that the approximation $\sum_{i,k}(1/A_{ik}) \approx \log h$ is correct to within an $O(\log\log h)$ factor, under the GRH. In any case, our primary justification for this heuristic is that it yields predictions that work well in practice.

Applying the heuristic yields

$$\sum_{i=1}^{m} \max_k \frac{1}{A_{ik}} = \sum_{i=1}^{m} \frac{1}{i} \approx \log m,$$

and

$$\max_i \left(\sum_{k=1}^{n} \frac{1}{A_{ik}} - \max_k \frac{1}{A_{ik}} \right) = \sum_{k=1}^{n-1} \frac{1}{mk+1} \approx \frac{\log n}{m},$$

TABLE 3. *Height bound optimization. (Note: Each row summarizes data collected for the first* 1000 *fundamental discriminants $|D| \geqslant N$. The value b_h is the unoptimized height bound, corresponding to $|G| = h$, while b_n is the optimized height bound, attained when $|G| = n$. Bars denote mean values.)*

N	$\bar{h}$	$\bar{n}$	$\bar{m}$	$\bar{b}_h$	$\bar{b}_n$	$\bar{b}_h/\bar{b}_n$	$\max b_h/b_n$
10^5	147	39	3.7	7626	4486	1.7	3.0
10^6	459	99	4.6	28387	14892	1.9	3.6
10^7	1470	254	5.8	105184	48667	2.2	4.2
10^8	4632	671	6.9	377174	157603	2.4	4.9
10^9	14640	1740	8.4	1339636	509688	2.6	5.5
10^{10}	46434	4849	9.6	4709013	1644023	2.9	5.6
10^{11}	146598	14777	9.9	16338099	5374105	3.0	6.1
10^{12}	462979	41189	11.2	56202741	17182753	3.3	6.6
10^{13}	1460465	114560	12.7	191932881	54720882	3.5	7.1
10^{14}	4644982	377059	12.3	656497242	179083436	3.7	7.4
10^{15}	14608895	964998	15.1	2211596515	560192565	4.0	9.2
10^{16}	46276481	2695634	17.2	7462636834	1742205583	4.3	9.2

[†]Under the random bisection model, a random integer N in some large interval will have prime-power factors whose logarithms approximate a geometric progression [4]. One then has divisors of N in most intervals of the form $[M, cM] \subset [1, N]$, for a suitable constant c. We heuristically assume that the same applies to h.

https://doi.org/10.1112/S1461157012001015 Published online by Cambridge University Press

which implies that b^* is within a constant factor of

$$\left(\log m + \frac{\log n}{m}\right)|D|^{1/2}.$$

This suggests that if we wish to minimize b^*, then we should make n exponentially larger than m. If we let $m \approx \log h$ and $n = h/m \approx h/\log h$, then we expect to have $b^* = O(|D|^{1/2} \log\log |D|)$, improving our worst-case bound by a factor of $\log |D|$, and improving the average case, where $\sum_{i,k}(1/A_{ik}) \approx \log h$, by a factor of $\log |D|/\log\ \log |D|$.

Using $m = \log h$ allows us to satisfy the bound (15) used to analyze Algorithm 2 whenever $\log q = O(|D|^{1/2})$, which is a very mild restriction. This choice of m precludes the improvement attained by Algorithm 2 under Proposition 1, since it makes n too big, but it does lead to the following claim.

Heuristic Claim 1. Assuming $\log q = O(|D|^c)$ for some $c < 1/2$, the average-case running time of both Algorithms 1 and 2 using $s = e_1$ is $O(|D|^{1/2} \log^{2+\epsilon} |D|)$.

Empirical data supporting the heuristic analysis above can be found in Table 3. Each row of the table gives data for 1000 fundamental discriminants of approximately the same size. We note that for the choice of G that minimizes b, the number of cosets m is quite close to $\log h$, on average, as expected. The two rightmost columns list, respectively, the average and best-case improvement achieved by optimizing the height bound. The growth rate is consistent with the $\Omega(\log h/\log\ \log h)$ prediction. In practice, the actual speedup is substantially better than the height-bound improvement would suggest, for reasons that will be explained in the next section where we analyze practical computations that amply demonstrate the benefit of optimizing the height bound.

6. *Computational results*

This section presents performance data and computational results. In order to handle a wider range of discriminants, and to give the most practically relevant examples, we use class invariants derived from various modular functions to which the CRT method has been adapted. These include, among others, the Weber ɟ-function, double η-quotients, and the Atkin functions A_N. We refer to [**24**, § 3] for definitions of these invariants and a detailed discussion of their implementation using the CRT method. Here we briefly summarize some key properties of the class invariants we use.

6.1. *Class invariants*

Let $\mathcal{O} = \mathbf{Z}[\tau]$ be an imaginary quadratic order with discriminant D, for some τ in the upper half-plane. The j-invariant $j(\tau)$ is a root of the Hilbert class polynomial H_D and generates the ring class field $K_\mathcal{O}$. Let $f(z)$ by a modular function of level N related to $j(z)$ by $\Psi_f(f(z), j(z)) = 0$, where $\Psi_f(F, J)$ is a polynomial with integer coefficients. The value $f(\tau)$ is an algebraic integer, and when $f(\tau)$ lies in $K_\mathcal{O}$ we call it a *class invariant*[†]. A given modular function typically yields class invariants only for a restricted subset of discriminants; for example, the primes dividing N must not be inert in $\mathbf{Q}(\sqrt{D})$.

We then define the *class polynomial* $H_D[f]$ by

$$H_D[f](X) = \prod_{\alpha \in \mathrm{cl}(\mathcal{O})} (X - [\alpha]f(\tau)).$$

[†]We do not require $f(\tau)$ to generate $K_\mathcal{O}$; we can obtain a generator as a root of $\Psi_f(f(\tau), Y)$.

https://doi.org/10.1112/S1461157012001015 Published online by Cambridge University Press

For the functions we consider, $H_D[f]$ has integer coefficients, and the techniques we have developed to find a root of $H_D \bmod q$ apply equally well to $H_D[f] \bmod q$. Having found a root f_0 of $H_D[f]$, we may obtain a root j_0 of H_D as a solution to $\psi(Y) = \Psi_f(f_0, Y) = 0$. Since the degree of ψ does not depend on D or q, we may bound it by $O(1)$, where the implicit constant depends on f. Thus deriving j_0 from f_0 takes just $O(\log^{2+\epsilon} q)$ time.

6.2. *Heuristic height bounds*

The key reason to consider alternative class invariants is that $H_D[f]$ may have much smaller coefficients than H_D. Let us define the *height factor* of f as $c(f) = \deg_F \Psi_f / \deg_J \Psi_f$. Asymptotically, we have

$$\mathrm{ht}(H_D[f]) = \frac{\mathrm{ht}(H_D)}{c(f)} + O(1),$$

where the constant $c(f)$ may be as large as 72. If b bounds the height of H_D, we regard $b/c(f)$ as an approximate bound on the height of $H_D[f]$, but add a small constant (say 256 bits) to account for the $O(1)$ term. We treat the optimized height bound b computed in § 5 in the same way.

This heuristic approach may, in rare cases, yield a bound that is too small. In practice this is easy to detect. The correct polynomial $V(Y)$ must split completely into linear factors in $\mathbf{F}_q[Y]$, and any sort of random error is extremely likely to yield a polynomial that does not. Verifying that $V(Y)$ splits into linear factors can easily be incorporated into the root-finding step at no additional cost[†]. If $V(Y)$ is found to be incorrect, we may then either retry with a larger height bound, or simply revert to $f = j$ and use the rigorous bound proven in § 5.

In most practical applications of the CM method, we seek an elliptic curve $E/\mathbf{F}_q$ with prescribed order N, where the prime factorization of N is known (or provisionally assumed). In this situation we can test whether we have constructed a suitable curve in time $O(\log^{2+\epsilon} q)$, via [**42**, Lemma 6], which is negligible. Although it is usually unnecessary, one can also verify the endomorphism ring of the constructed curve, provided that we know the factorization of the integer v in the norm equation $4q = t^2 - v^2 D$. Using the algorithm in [**11**, Algorithm 2], this takes time subexponential in $\log |D|$, under heuristic assumptions, which is also negligible.

6.3. *Implementation*

Our tests were performed on a small network of quad-core AMD Phenom II 945 CPUs, each clocked at 3.0 GHz. The computation of class polynomials (or decompositions thereof) was distributed across up to 48 cores, depending on the size of the test, with essentially linear speedup, while all root-finding operations were performed on a single core. For consistency we report total CPU times, summed over all threads.

The software was implemented using the `gmp` [**27**] and `zn_poly` [**30**] libraries, with the `gcc` compiler [**25**]. Polynomial arithmetic modulo the small primes $p \in S$ was handled via `zn_poly`, while polynomial arithmetic modulo large primes q used the cache-friendly truncated FFT approach described in [**31**], layered on top of the `gmp` library. In order to simplify the implementation, when selecting the subgroup G to optimize the height bound, only subgroups of the form (9) in § 2.6 were considered. Additionally, of the various space optimizations described in § 4.3 that may be applied to Algorithm 2, only the changes necessary to achieve a space complexity of $O(h \log h + (m+n) \log q)$ were used (see § 4.3.2). A more complete implementation would improve some of the results presented here.

[†]The first step of the standard root-finding procedure computes the polynomial $\gcd(Y^q - Y, V(Y))$ whose degree is the number of distinct roots of V; duplicate roots can be accounted for by taking gcds with derivatives of V.

https://doi.org/10.1112/S1461157012001015 Published online by Cambridge University Press

As noted in Remark 2, in our implementation we fixed $s = e_1$. This choice of s worked in every large ($|D| > 10^6$, $\log q > 160$) example that we tested, which included more than a million different combinations of D and G. We conjecture that $s = e_1$ always works when using j-invariants, but note that it can fail for other class invariants in rare cases (the handful of exceptions we found all involved very small discriminants, and in each such case switching to $s = e_2$ worked).

6.4. *Accelerated CM computations with Algorithm 1*

We applied Algorithm 1 to several examples that have previously appeared in the literature. The examples in Table 4 are taken from [**42**, Table 2] where they appear as representatives of a large set of computations to construct elliptic curves suitable for pairing-based cryptography. These examples are also used in [**24**, Table 1] with the class invariants we use here[†]. The first five discriminants in Table 5 originally appeared in [**21**, Table 1], and can also be found in [**42**, Table 4] and [**24**, Table 2]. The remaining discriminants are from [**42**, Table 4].

The time T_{poly} listed in Tables 4–7 is the total time spent computing the polynomial V and the polynomials W_k, in the case of Algorithm 1 (Steps 1–6), and the total time spent computing the polynomial V and the values w_k in the case of Algorithm 2 (steps 1–5 and 7–8), including all precomputation. The time T_{root} is the time spent on root-finding operations (steps 7–8 in Algorithm 1 and steps 6 and 9 in Algorithm 2). For the smaller examples, these are averages over 10 runs; with the large examples there is very little variance in the root-finding times.

The '**Standard**' computations listed in Tables 4 and 5 correspond directly to the computations in [**24**], and are equivalent to running Algorithm 1 with $G = \text{cl}(\mathcal{O})$. The '**Accelerated**' computations used Algorithm 1 with G chosen to minimize T_{tot}, based on heuristic formulas for T_{poly} and T_{root} extrapolated from empirical data. In most cases this minimizes the corresponding height bound, but not always; in the $h = 100000$ example of Table 5, using $n = 5000$ rather than $n = 2500$ improves the ratio b_h/b_n from 4.93 to 5.84 and reduces T_{poly} by 10 s, but it increases T_{root} by 21 s, so this subgroup was not chosen.

The T_{poly} times listed in Table 4 are in each case slightly greater than the quantity $|S|(T_{\text{find}} + T_{\text{enum}} + T_{\text{build}})$, due to time spent updating the CRT data in step 3e of Algorithm 1, which is included in T_{poly}. This difference is only a few percent for the values of q used in these examples, but becomes more significant when q is very large (see § 6.5).

The third example in Table 4 illustrates four ways in which Algorithm 1 can reduce the time required to apply the CM method using the CRT approach.

(1) The height bound $b_n = 50180$ is about 3 times smaller than $b_h = 151359$, which reduces $|S|$ similarly, from 4477 to 1519.

(2) The average time T_{find} spent finding an element of $\text{Ell}_{\mathcal{O}}(\mathbf{F}_p)$ is reduced from 42 to 28 ms, because the primes that remain in S are those for which it is easier to find curves in $\text{Ell}_{\mathcal{O}}(\mathbf{F}_p)$.

(3) The average time T_{build} spent building polynomials from their roots (or computing linear combinations) is reduced from 63 to 22 ms, because the degrees of the polynomials involved are $m = 22$ and $n = 2458$ rather than $h = 54076$.

(4) Working with polynomials of lower degree reduces the time T_{root} spent finding roots dramatically: from 171 s to 5 s.

As may be seen in Tables 4 and 5, the speedup achieved by Algorithm 1 is typically better than the height bound ratio b_h/b_n, for the reasons noted above. In the last example of Table 5, with discriminant $D = -170868609071$ and class number $h = 1000000$, computing an optimized

[†]The timings listed here for the standard computations are slightly better (about 5%) than those in [**24**] due to a more recent version of `gmp`.

https://doi.org/10.1112/S1461157012001015 Published online by Cambridge University Press

height bound b_n with $n = 25000$ improves the height bound by a factor of about 7, but T_{poly} is reduced by nearly a factor of 14 and T_{root} is reduced even more.

The discriminant $D = -170868609071$ also appears in [42, Table 4], which lists a time equivalent to 150 CPU days on our current test platform to compute the Hilbert class polynomial H_D modulo a 256 bit prime q. Here we instead use the Weber ɟ-function, with a height factor of 72, and are able to further improve the height bound by a further factor

TABLE 4. *Example CM constructions.*

	Example 1	Example 2	Example 3
Discriminant $\lvert D\rvert$	13569850003	11039933587	12901800539
Field size $\lceil \lg q \rceil$	177	231	172
Class number h	20203	11280	54076
Presentation $\ell_1^{r_1}, \ldots, \ell_k^{r_k}$	7^{20203}	$17^{1128}, 19^{10}$	$3^{27038}, 5^2$
Modular function f	A_{71}	A_{47}	A_{71}
Height factor $c(f)$	36	24	36
Standard			
Subgroup size $\lvert G\rvert = h$	20203	11280	54706
Height bound b_h	63127	56631	151939
Number of primes $\lvert S\rvert$	1993	1783	4477
T_{find} (ms)	48	110	42
T_{enum} (ms)	33	48	23
T_{build} (ms)	15	7	63
T_{poly} (s)	197	295	597
T_{root} (s)	56	54	171
T_{tot} (s)	**253**	**347**	**768**
Accelerated			
Subgroup size $\lvert G\rvert = n$	227	1128	2458
Height bound b_n	35115	30957	50180
Number of primes $\lvert S\rvert$	1115	994	1519
T_{find} (ms)	44	105	28
T_{enum} (ms)	33	47	23
T_{build} (ms)	6	3	22
T_{poly} (s)	95	155	118
T_{root} (s)	0	4	5
T_{tot} (s)	**95**	**159**	**123**

TABLE 5. *CM constructions using the Weber ɟ-function and* $q \approx 2^{256}$.

				Standard		Accelerated	
$\lvert D\rvert$	h	n	b_h/b_n	T_{poly}	T_{root}	T_{poly}	T_{root}
6961631	5000	250	3.63	1.0	25	0.2	0.7
23512271	10000	250	3.65	3.9	58	0.8	0.7
98016239	20000	625	4.10	21	126	3.5	2.1
357116231	40000	625	4.82	90	282	11	2.2
2093236031	100000	2500	4.93	750	812	88	11
8364609959	200000	4000	6.34	3590	1805	301	16
17131564271	300000	6250	6.47	9070	2890	708	19
30541342079	400000	12500	6.31	16900	3910	1380	71
42905564831	500000	15625	7.11	28300	4410	2300	85
170868609071	1000000	25000	7.06	123000	9260	8840	159

https://doi.org/10.1112/S1461157012001015 Published online by Cambridge University Press

of 7 using a decomposition of the class polynomial $H_D[\mathfrak{f}]$. We eventually obtain a root of the original polynomial $H_D \bmod q$, and it takes only 2.5 CPU hours to do so, an overall speedup by nearly a factor of 1500.

6.5. *Optimizing space with Algorithm 2*

As noted in Remark 2, choosing G to optimize the height bound may negate any performance advantage Algorithm 2 might have over Algorithm 1. Indeed, Algorithm 1 is usually faster, due to the larger height bound required by Algorithm 2, and the fact that Algorithm 2 repeats the enumeration step in its second stage. However when q is large, Algorithm 2 may use much less space than Algorithm 1, which can actually lead to a better running time. In this scenario we use the modified form of Algorithm 2 described in § 4.4, which makes the height bound increase negligible, and to optimize space we choose G so that $m = h/n$ is approximately equal to but no larger than $n = |G|$.

Table 6 compares the time and space required by Algorithms 1 and 2 for a fixed discriminant $D = -300000504611$ and increasingly large primes $q \approx 2^k$. The class number is $h = 2^{18}$, and in each case we choose G so that $m = n = 2^9$. In addition to the times T_{find}, T_{enum}, and T_{build} listed in Table 4, we also list the average time T_{crt} spent updating CRT data for each of the primes $p \in S$. This time is negligible when q is of moderate to cryptographic size (say, up to 1024 bits), but when q is very large, as may occur in elliptic curve primality proving [**3, 36**], the time Algorithm 1 spends updating its CRT data becomes quite significant[†].

TABLE 6. *Algorithms 1 and 2 with* $n = |G| = 512$ *and* $q \approx 2^k$. *(Note:* $D = -300000504611$ *with* $h(D) = 262144$ *using* A_{71}.*)*

lg k	alg	b (bits)	$\lvert S\rvert$	T_{find} (ms)	T_{enum} (ms)	T_{build} (ms)	T_{crt} (ms)	T_{poly} (s)	T_{root} (s)	CRT data (MB)
7	1	378315	10013	165	107	166	25	4660	1	8.2
	2	378452	10016	166	213	168	0	5510	1	0.03
8	1	378315	10013	165	107	166	26	4660	3	12.6
	2	378350	10020	166	213	169	0	5510	3	0.05
9	1	378315	10013	165	107	165	28	4670	12	21.0
	2	378836	10026	166	213	169	0	5520	12	0.08
10	1	378315	10013	165	107	166	33	4270	37	37.7
	2	379348	10039	166	213	169	0	5530	37	0.15
11	1	378315	10013	165	107	166	43	4820	142	71.3
	2	380372	10066	166	213	169	0	5540	142	0.28
12	1	378315	10013	165	107	166	73	5120	697	138
	2	382420	10119	166	213	169	0	5590	697	0.54
13	1	378315	10013	165	107	166	129	5690	3420	273
	2	386516	10225	167	213	169	0	5630	3420	1.06
14	1	378315	10013	165	107	166	225	6700	16510	541
	2	394708	10437	168	214	170	1	5810	16510	2.11
15	1	378315	10013	165	107	166	461	9100	81100	1078
	2	411902	10859	170	214	171	2	6060	81100	4.21

[†]As discussed in [**42**, § 6.3], we should eventually transition from the explicit CRT to a standard CRT approach as q grows, but here lg q is still much smaller than the height bound b.

https://doi.org/10.1112/S1461157012001015 Published online by Cambridge University Press

One can see the two disadvantages of Algorithm 2 in Table 6; it requires a slightly larger S, and T_{enum} is doubled. However Algorithm 2 needs much less space for its CRT data, and spends negligible time updating it. In our implementation both Algorithms 1 and 2 use $O(h \log |D|)$ space for the computations performed modulo each prime $p \in S$, about 10 MB in this example, but Algorithm 1 requires $O(h \log q)$ space for its CRT data, regardless of the choice of G, whereas Algorithm 2 only requires $O((m+n) \log q)$ space. As shown in the last two rows of Table 6, for $q \approx 2^{32768}$ Algorithm 1 uses more than 1 GB of CRT data, compared to about 4 MB for Algorithm 2, which leads to a significant time advantage for Algorithm 2. Note that the memory required by Algorithm 2 to store its CRT data is actually half the size listed in Table 6, since with $m = n$ the CRT data is evenly split across the 2 stages and the CRT data for the first stage can be discarded before the second stage begins.

Due to its superior space complexity, Algorithm 2 is able to effectively handle a broader range of $|D|$ and q than Algorithm 1. The next section gives an example of a computation with $|D| \approx 10^{15}$ and $q \approx 10^{10000}$ that is easily handled by Algorithm 2 but would be impractical to compute on our test platform using Algorithm 1, or any algorithm that requires space proportional to the size of $H_D \bmod q$.

6.6. *Some large examples*

We also tested Algorithms 1 and 2 with some larger discriminants, beginning with $D = -1000000013079299$, which has class number $h(D) = 10034174$. This discriminant appears in [**24**], where it was used to construct an elliptic curve over a 256 bit prime field via a class invariant derived from the Atkin function A_{71}. As noted in [**24**], this set of parameters was chosen so that the level $N = 71$ is ramified in $\mathbf{Q}(\sqrt{D})$, which allows us to work with the square root of the class polynomial $H_D[A_{71}]$, reducing both the degree and the height bound by a factor of two. The decomposition techniques described here can be applied directly to the polynomial $\sqrt{H_D[A_{71}]}$, allowing both Algorithms 1 and 2 to take advantage of this situation.

Table 7 gives timings for five computations that constructed elliptic curves modulo a 256 bit prime q by obtaining a root of the polynomial $\sqrt{H_D[A_{71}]} \bmod q$. The first row corresponds to the original computation in [**24**]. The next two rows give timings for Algorithms 1 and 2 when the subgroup G is chosen to optimize the running time of Algorithm 1, with $n = |G| = 44399$. This reduced the total CPU time by nearly a factor of 5, allowing the entire computation to be completed in less than a day of elapsed time on 48 cores. The portion of CPU time spent on root-finding was cut dramatically, from more than a day to under five minutes. This improvement is particularly helpful in a distributed implementation, as root-finding is not as easy to parallelize as the other steps and is most conveniently performed on a single CPU.

The last two rows of Table 7 give timings for Algorithms 1 and 2 when G is chosen to optimize the space used by Algorithm 2. This increases the running time by about 15%, but requires

TABLE 7. *CM computations with* $|D| = 10^{15} + 13079299$ *and* $q \approx 2^{256}$.

alg	n	b (bits)	$\lvert S\rvert$	T_{find} (ms)	T_{enum} (ms)	T_{build} (ms)	T_{crt} (ms)	T_{poly} (s)	T_{root} (s)
–	–	21533401	438700	17500	1580	25000	531	19400000	95600
1	44399	8315747	170112	12700	1580	9210	531	4120000	237
2	44399	8344202	150662	12700	3160	8350	5	4190000	237
1	3277	11130011	227504	13700	1580	7180	535	5260000	0
2	3277	11518641	235482	14400	3170	2510	0	4780000	0

https://doi.org/10.1112/S1461157012001015 Published online by Cambridge University Press

less than 2 MB of CRT data, compared to about 250 MB for the original computation (and Algorithm 1). This reduced the total memory usage from around 500 MB to about 100 MB.

As noted in § 6.5, the reduced space required by Algorithm 2 becomes critical for larger values of q. To demonstrate this, we performed a sixth computation with the discriminant $D = -1000000013079299$, this time using $q \approx 2^{33220}$. The total running time for Algorithm 2 was about 5800000 s (including root-finding), just a 20% increase over the 256 bit computation, and the size of the CRT data was about 25 MB, yielding a total memory usage under 200 MB. The 10000-digit prime q and the coefficients of the constructed curve are too large to conveniently print here, but they are available at http://math.mit.edu/~drew.

By contrast, Algorithm 1, and the algorithm of [**24**], requires more than 20 GB of CRT data for this example, and this data needs to be updated for every prime $p \in S$. This makes it infeasible to even attempt this computation with Algorithm 1 on our test platform, whereas Algorithm 2 was easily able to address this example.

Finally, we performed two record-setting computations, one with $h(D) > 5 \cdot 10^7$ and the other with $|D| > 10^{16}$, again using the polynomial $\sqrt{H_D[A_{71}]}$. First, we used the discriminant $D = -506112046263599$ with class number $h(D) = 50666940$ to construct an Edwards curve of the form $x^2 + y^2 = 1 + cx^2y^2$, where

$$c = 34995650161014075667740469266710958774247253260831352020801431139436365125 45,$$

over the 256 bit prime field $\mathbf{F}_q$ with

$$q = 28948022309329048855892746252171986268338819619472424415843054443714437912893.$$

The trace of this curve is

$$t = 340282366920938463463374607431768266146,$$

and the group order $q + 1 - t$ is 4 times a prime. This computation took approximately 200 days of CPU time (about 5 days elapsed time) using Algorithm 2, which in this case was faster than Algorithm 1.

Next we used $D = -10000006055889179$ with class number $h(D) = 25459680$ to construct an elliptic curve with Weierstrass equation $y^2 = x^3 - 3x + c$, where

$$c = 153252523848878822277574217481027943183495187127094873898179059292390075686 05,$$

over the 256 bit prime field $\mathbf{F}_q$ with

$$q = 28948022309329048855892746252171992875431396939874100252456123922623314798263.$$

This curve has trace

$$t = -340282366920938463463374607431768304979,$$

and the group order is prime. This computation took about 400 days of CPU time (under 8 days elapsed time) using Algorithm 1, which was faster than Algorithm 2 for this discriminant.

Acknowledgement. I am grateful to Andreas Enge and François Morain for providing further details of the algorithms in [**23, 28**] and to David Harvey for his assistance with `zn_poly`. I would also like to sincerely thank the anonymous referee, whose careful reading and comprehensive feedback greatly improved the clarity and rigor of this article.

References

1. A. Agashe, K. Lauter and R. 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*, Fields Institute Communications 41 (eds A. J. van der Poorten and A. Stein; American Mathematical Society, 2004) 1–17.

2. M. Agrawal, N. Kayal and N. Saxena, 'PRIMES is in P', *Ann. Math.* (2) 160 (2004) 781–793.

https://doi.org/10.1112/S1461157012001015 Published online by Cambridge University Press

3. A. O. L. Atkin and F. Morain, 'Elliptic curves and primality proving', *Math. Comp.* 61 (1993) 29–68.
4. E. Bach, 'Analytic methods in the analysis and design of number-theoretic algorithms', *ACM Distinguished Dissertation 1984* (MIT Press, 1985).
5. E. Bach, 'Explicit bounds for primality testing and related problems', *Math. Comp.* 55 (1990) no. 191, 355–380.
6. S. Baier and L. Zhao, 'On primes in arithmetic progressions', *Int. J. Number Theory* 5 (2009) no. 6, 1017–1035.
7. J. Belding, R. Bröker, A. Enge and K. Lauter, 'Computing Hilbert class polynomials', *Algorithmic Number Theory Symposium–ANTS VIII*, Lecture Notes in Computer Science 5011 (eds A. J. van der Poorten and A. Stein; Springer, 2008) 282–295.
8. E. R. Berlekamp, 'Factoring polynomials over large finite fields', *Math. Comp.* 24 (1970) no. 111, 713–735.
9. D. J. Bernstein, '*Detecting perfect powers in essentially linear time, and other studies in computational number theory*', PhD Thesis, University of California at Berkeley, 1995.
10. D. J. Bernstein and J. P. Sorenson, 'Modular exponentiation via the explicit Chinese Remainder theorem', *Math. Comp.* 76 (2007) 443–454.
11. G. Bisson and A. V. Sutherland, 'Computing the endomorphism ring of an ordinary elliptic curve over a finite field', *J. Number Theory* 113 (2011) 815–831.
12. R. Bröker, 'A p-adic algorithm to compute the Hilbert class polynomial', *Math. Comp.* 77 (2008) 2417–2435.
13. R. Bröker, K. Lauter and A. V. Sutherland, 'Modular polynomials via isogeny volcanoes', *Math. Comp.* 81 (2012) 1201–1231.
14. J. Buchmann and U. Vollmer, *Binary quadratic forms: an algorithmic approach*, Algorithms and Computations in Mathematics 20 (Springer, Berlin, 2007).
15. J. Chao, O. Nakamura, K. Sobataka and S. Tsujii, 'Construction of secure elliptic cryptosystems using CM tests and liftings', *Advances in cryptology–ASIACRYPT'98*, Lecture Notes in Computer Science 1514 (Springer, 1998) 95–109.
16. A. M. Childs, D. Jao and V. Soukharev, *Constructing elliptic curve isogenies in quantum subexponential time*, Preprint, 2011, http://arxiv.org/abs/1012.4019v2.
17. H. Cohen and H. W. Lenstra Jr., 'Heuristics on class groups of number fields', *Number Theory, Noordwijkerhout 1983*, Lecture Notes in Mathematics 1068 (Springer, 1984) 33–62.
18. J.-M. Couveignes and T. Henocq, 'Action of modular correspondences around CM points', *Algorithmic Number Theory Symposium–ANTS V*, Lecture Notes in Computer Science 2369 (eds C. Fieker and D. R. Kohel; Springer, 2002) 234–243.
19. D. A. Cox, *Primes of the form $x^2 + ny^2$: Fermat, class field theory, and complex multiplication* (John Wiley and Sons, 1989).
20. R. Crandall and C. Pomerance, *Prime numbers: a computational perspective*, 2nd edn (Springer, 2005).
21. A. Enge, 'The complexity of class polynomial computation via floating point approximations', *Math. Comp.* 78 (2009) 1089–1107.
22. A. Enge and F. Morain, 'Comparing invariants for class fields of imaginary quadratic fields', *Algorithmic Number Theory Symposium–ANTS V*, Lecture Notes in Computer Science 2369 (eds C. Fieker and D. R. Kohel; Springer, 2002) 252–266.
23. A. Enge and F. Morain, 'Fast decomposition of polynomials with known Galois group', *Applied algebra, algebraic algorithms, and error correcting codes — 2003*, Lecture Notes in Computer Science 2643 (Springer, 2003) 254–264.
24. A. Enge and A. V. Sutherland, 'Class invariants for the CRT method', *Algorithmic Number Theory Symposium–ANTS IX*, Lecture Notes in Computer Science 6197 (eds G. Hanrot, F. Morain and E. Thomé; Springer, 2010) 142–156.
25. Free software foundation, '*GNU compiler collection*', version 4.4.3, 2010, available at http://gcc.gnu.org/.
26. A. Gee and P. Stevenhagen, 'Generating class fields with Shimura reciprocity', *Algorithmic Number Theory Symposium–ANTS III*, Lecture Notes in Computer Science 1423 (Springer, 1998) 442–453.
27. T. Granlund *et al.*, *GNU* multiple precision arithmetic library, September 2010, version 5.0.1, available at http://gmplib.org/.
28. G. Hanrot and F. Morain, 'Solvability by radicals from an algorithmic point of view', *International Conference on Symbolic and Algebraic Computation–ISSAC 2001* (ACM, 2001) 175–182.
29. G. H. Hardy and E. M. Wright, *An introduction to the theory of numbers*, 5th edn (Oxford Science Publications, 1979).
30. D. Harvey, *zn_poly*: *a library for polynomial arithmetic*, version 0.9, 2008, http://cims.nyu.edu/∼harvey/zn_poly.
31. D. Harvey, 'A cache-friendly truncated FFT', *Theoret. Comput. Sci.* 410 (2009) 2649–2658.
32. S. Ionica and A. Joux, 'Pairing the volcano', *Algorithmic Number Theory Symposium–ANTS IX*, Lecture Notes in Computer Science 6197 (eds G. Hanrot, F. Morain and E. Thomé; Springer, 2010) 201–218.
33. 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, Durham, 1975)* (Academic Press, 1977) 409–464.
34. S. Lang, *Elliptic functions*, 2nd edn (Springer, 1987).
35. J. E. Littlewood, 'On the class-number of the corpus $P(\sqrt{-k})$', *Proc. Lond. Math. Soc.* 27 (1928) 358–372.

36. F. Morain, 'Primality proving using elliptic curves: an update', *Algorithmic Number Theory Symposium–ANTS III*, Lecture Notes in Computer Science 1423 (Springer, 1998) 111–127.
37. K. Rubin and A. Silverberg, 'Choosing the correct elliptic curve in the CM method', *Math. Comp.* 79 (2010) 545–561.
38. A. Schönhage, 'Fast reduction and composition of binary quadratic forms', *International Symposium on Symbolic and Algebraic Computation–ISSAC'91* (ed. S. M. Watt; ACM, 1991) 128–133.
39. A. Schönhage and V. Strassen, 'Schnelle Multiplikation großer zahlen', *Computing* 7 (1971) 281–292.
40. J.-P. Serre, 'Complex multiplication', *Algebraic number theory* (eds J.W.S. Cassels and A. Fröhlich; Academic Press, 1967).
41. A. V. Sutherland, '*Order computations in generic groups*', PhD Thesis, MIT, 2007, http://groups.csail.mit.edu/cis/theses/sutherland-phd.pdf.
42. A. V. Sutherland, 'Computing Hilbert class polynomials with the Chinese remainder theorem', *Math. Computation* 80 (2011) 501–538.
43. A. V. Sutherland, 'Structure computation and discrete logarithms in finite abelian p-groups', *Math. Comp.* 80 (2011) 477–500.
44. B. Leendert and van der Waerden, *Algebra*, vol. I (Springer, 1991). Originally published in German as *Moderne algebra* in 1930–1931.
45. J. von zur Gathen and J. Gerhard, *Modern computer algebra*, 2nd edn (Cambridge University Press, 2003).
46. H. Weber, *Lehrbuch der algebra*, 3rd edn, vol. III (Chelsea, 1961).

Andrew V. Sutherland
Department of Mathematics
Massachusetts Institute of Technology
77 Massachusetts Avenue, Cambridge,
MA 02139
USA

drew@math.mit.edu