CONSTRUCTING ELLIPTIC CURVES WITH A KNOWN NUMBER OF POINTS OVER A PRIME FIELD

AMOD AGASHE, KRISTIN LAUTER, AND RAMARATHNAM VENKATESAN

ABSTRACT. Elliptic curves with a known number of points over a given prime field $\mathbf{F}_n$ are often needed for use in cryptography. In the context of primality proving, Atkin and Morain suggested the use of the theory of complex multiplication to construct such curves. One of the steps in this method is the calculation of a root modulo n of the Hilbert class polynomial $H_D(X)$ for a fundamental discriminant D. The usual way of doing this calculation is to first compute $H_D(X)$ over the integers and then to find the root modulo n. We present a modified version of the Chinese remainder theorem to compute $H_D(X)$ modulo n directly from the knowledge of $H_D(X)$ modulo enough small primes. Our heuristic complexity analysis suggests that asymptotically our algorithm is an improvement over previously known methods.

1. INTRODUCTION

In order to use elliptic curves in cryptography, one often needs to construct elliptic curves with a known number of points over a given prime field. One way of doing this is to randomly pick elliptic curves and then to count the number of points on the curve over the prime field, repeating this until the desired number of points is found. Atkin and Morain [AtMor] pointed out that instead, one can use the theory of complex multiplication to construct elliptic curves with a known number of points. Although at present it may still be more efficient to count points on random curves, we hope that improving the complex multiplication method will eventually yield a more efficient algorithm. In some situations, using complex multiplication methods is the only practical possibility (e.g., if the prime is too large for point-counting to be efficient yet the discriminant of the imaginary quadratic field is relatively small). This paper provides a new version of the complex multiplication method.

Suppose n is an integer, usually a prime or a pseudo-prime, and one wants to construct an elliptic curve modulo n such that the number of points on that curve modulo n is known. One of the steps in the complex multiplication method for constructing such an elliptic curve is the calculation of the Hilbert class polynomial $H_D(X)$ modulo n for a certain fundamental discriminant D. The usual way to do this is to compute $H_D(X)$ over the integers and then to reduce it modulo n. Atkin and Morain proposed computing $H_D(X)$ as an integral polynomial by listing all the relevant binary quadratic forms, associating to each form an algebraic integer, evaluating the j-function at each of these integers as a floating point integer with sufficient precision, and then taking the product and rounding the coefficients to nearest integers. Let $d = |D|$. If we use the bound given by formula (3), then in view of [LL, §5.10], the computation of $H_D(X)$ by this method takes time $O(d^2(\log d)^2)$.

In [CNST, §4], the authors suggest that one can compute $H_D(X) \bmod p$ for sufficiently many small primes p and then use the Chinese remainder theorem (CRT) to compute $H_D(X)$ as a polynomial with integer coefficients. In this paper we use a modified version of CRT to compute $H_D(X)$ modulo n directly (knowing $H_D(X) \bmod p$ for sufficiently many

2000 *Mathematics Subject Classification.* 14H52, 11G15, 11G20, 11Y16, 11Z05.

Key words and phrases. elliptic curves, complex multiplication, finite fields, Chinese remainder theorem.

We would like to thank the anonymous referees of an earlier version of the paper for pointing out mistakes and making suggestions. The first author would like to thank D. Kohel, H. Lenstra, F. Morain and J. Vaaler for several useful conversations, and M.S.R.I., I.H.E.S., and M.P.I. for their generous hospitality. The second author would like to thank R. Schoof for pointing out the paper [CNST], and P. Montgomery, D. Bernstein, A. Stein, and H. Cohn for useful discussions.

small primes p), *without* computing the coefficients of $H_D(X)$ as integers. We also give the mathematical justification and details of the (usual) CRT method, which were omitted in [CNST, §4] and also correct their erroneous complexity analysis. By avoiding the computation of the coefficients of $H_D(X)$ as integers, we obtain an algorithm with asymptotically shorter running time as d gets large. Also, both CRT approaches require less precision of computation than the Atkin-Morain approach.

Our complexity analysis in Section 3 shows that, when d is large, with high probability, the running time of one of the versions of our algorithm is

$$\mathrm{O}(d^{3/2}(\log d)^{10} + d(\log d)^2 \log n + \sqrt{d}(\log n)^2),$$

which is better than the Atkin-Morain method when d is sufficiently large (roughly speaking, bigger than $(\log n)^2$). Our algorithm has one step in common with the (usual) CRT method; this step takes time $\mathrm{O}(d^{3/2}(\log d)^{10})$. For the only other step, our algorithm takes time

$$\mathrm{O}(d(\log d)^4 + d(\log d)^2 \log n + \sqrt{d}(\log n)^2),$$

while the (usual) CRT method takes time

$$\mathrm{O}(d(\log d)^2 \log n + d^{3/2}(\log d)^4).$$

Thus we obtain an improvement over the (usual) CRT method when d is greater than $(\log n)^2$.

Note that in [AtMor], the authors mention that using Weber polynomials works better in practice than using Hilbert polynomials, as they have smaller coefficients. At the moment, we do not have a generalization of our algorithm which works with Weber polynomials. The use of Weber polynomials only reduces the number of digits in the coefficients by a constant (see [Cohen, p.409]), hence will only change the time taken by a constant factor independent of d; so the asymptotic complexity estimates remain the same.

Note that we focus only on one step of the complex multiplication algorithm to construct an elliptic curve with known number of points over a prime field, viz., the computation of the Hilbert class polynomial, which is a very time consuming step. The other time consuming step is the computation of a root of $H_D(X)$ modulo n, which (by [LL, §5.10]) takes time $\mathrm{O}(d(\log n)^3)$. Which of these two steps will dominate (when we use our algorithm to compute $H_D(X)$ modulo n) depends on the relative values of d and n.

It is not clear how our method compares to existing methods computationally. While we did some examples (reported in Section 6), they involved small discriminants, where existing methods are already very fast. The purpose of this paper is to suggest a new version of the complex multiplication method and to present a complexity analysis, leaving the task of efficient implementation for the future.

The paper is organized as follows: in Section 2, we give a brief description of the complex multiplication method for generating elliptic curves. In Section 3, we give an outline of our algorithm and discuss its complexity. In Sections 4 and 5, we explain the details of some of the steps of the algorithm. Finally in Section 6, we give some computational examples of our method.

2. Complex multiplication method

Suppose n is an integer, usually a prime or a pseudo-prime, and one wants an elliptic curve modulo n such that the number of points on that curve modulo n is known (if n is not known to be a prime, then we just consider a Wierstrass equation modulo n). We briefly review the complex multiplication method for constructing such an elliptic curve, referring the reader to [AtMor] and [BSS, Chap. VII] for details.

We first discuss the situation where the number of points on the elliptic curve is a prespecified non-negative integer N in the Hasse-Weil interval $[n+1-2\sqrt{n}, n+1+2\sqrt{n}]$. Suppose E is an elliptic curve over $\mathbf{F}_n$ with N points over $\mathbf{F}_n$. Now $\#E(\mathbf{F}_n) = n+1-t'$, where t' is the trace of the Frobenius endomorphism of E over $\mathbf{F}_n$. With the above in mind, set $t = n+1-N$ and let

$$D = t^2 - 4n.$$

Then the Frobenius endomorphism of E has characteristic polynomial $x^2 - tx + p$, whose roots lie in $\mathbf{Q}(\sqrt{D})$. It is standard to associate the Frobenius endomorphism with a root of the characteristic polynomial above. Let R denote the endomorphism ring of E. Suppose $t \neq 0$, so that E is not supersingular (by [Silv1, Ex. 5.10b]), and suppose that D is not a perfect square. Then R is an order in the ring of integers of $K = \mathbf{Q}(\sqrt{D})$ ([Silv1, Thm 3.1.b]).

Recall that a negative integer D' is said to be a *fundamental discriminant* if it is not divisible by any square of an odd prime and satisfies $D' \equiv 1 \bmod 4$ or $D' \equiv 8, 12 \bmod 16$ (i.e., D' is the discriminant of a quadratic imaginary field). If D is a fundamental discriminant, then R is automatically equal to $\mathcal{O}_K$, since then the Frobenius endomorphism generates the full ring of integers and is contained in the endomorphism ring. Our results can be generalized to orders in the ring of integers, but the algorithm will become more complicated. So suppose now that $t \neq 0$, and D is a fundamental discriminant (which implies in particular that D is not a perfect square).

The *Hilbert class polynomial* $H_D(X)$ is defined as:

$$H_D(X) = \prod \left(X - j\left(\frac{-b + \sqrt{D}}{2a} \right) \right), \tag{1}$$

where the product ranges over the set of $(a, b) \in \mathbf{Z} \times \mathbf{Z}$ such that $ax^2 + bxy + cy^2$ is a primitive, reduced, positive definite binary quadratic form of discriminant D for some $c \in \mathbf{Z}$, and j denotes the modular invariant. The degree of $H_D(X)$ is equal to h, the class number of $\mathcal{O}_K$. It is known that $H_D(X)$ has integer coefficients. The equivalence between isomorphism classes of elliptic curves over $\mathbf{C}$ with endomorphism ring equal to $\mathcal{O}_K$ and primitive, reduced, positive definite binary quadratic forms of discriminant D allows us to interpret a root of this polynomial as the j-invariant of an elliptic curve having this endomorphism ring. Since our goal is to find such an elliptic curve modulo n, it suffices to find a root j of $H_D(X)$ modulo n.

If $j \neq 0, 1728$, consider the elliptic curve with Weierstrass equation (assume $n \neq 2, 3$)

$$y^2 = x^3 + 3kx + 2k,$$

where

$$k = \frac{j}{1728 - j}.$$

The number of points on this elliptic curve is either $n+1+t$ or $n+1-t$, and one can easily check which one it is by raising randomly chosen points to one of the possible group orders. If the number of points is $n + 1 - t = N$, then we have an elliptic curve with the desired number of points; if not, we just take the quadratic twist, which has $n + 1 - t = N$ points. If $j = 0$ or $j = 1728$, then the situation is more complicated; in practice one usually would like to avoid such special j-invariants in any case.

We now mention how the discussion above changes if the number of points is not prespecified. If there is no restriction at all on the number of points, then, given n, we are free to to choose any negative fundamental discriminant D that satisfies $4n = x^2 - Dy^2$ for some integers x and y with $x \neq 0$. With this value of D, we just perform the steps mentioned in the two paragraphs just above. This gives us an elliptic curve whose number of points over $\mathbf{F}_n$ is either $n + 1 + x$ or $n + 1 - x$, and again, one can easily check which one it is by raising randomly chosen points to one of the possible group orders. If in addition the number of points is required to satisfy certain other properties (e.g., that it has a "large" prime factor as in primality proving [AtMor]), then that puts more restrictions on D (by putting restrictions on x).

3. Our algorithm and its complexity

3.1. **Overview of the algorithm.** As before, let D be a fundamental discriminant and let $d = |D|$. Let $K = \mathbf{Q}(\sqrt{D})$ and let $\mathcal{O}_K$ denote the ring of integers of K. Let h denote the class number of $\mathcal{O}_K$ and let B be the upper bound on the size of the coefficients of $H_D(X)$

given by formula (2) in Section 3.3. Let n be a given prime number (or more generally, a pseudoprime).

Here is our algorithm for computing $H_D(X) \bmod n$; it comes in two versions, Version A and Version B, which differ only in Step (1) below:

Step (0) Compute h using any of the standard algorithms (e.g., see [Cohen, §5.4]) and compute B using formula (2) in Section 3.3. Fix a small real number $\epsilon > 0$ (e.g., $\epsilon = 0.001$), and let $M = B/(1/2 - \epsilon)$.

Step (1) Compute $H_D(X)$ modulo sufficiently many small primes:

Version A: This can be used whenever $d \not\equiv 7 \bmod 8$, and is probabilistic.

(a) Generate a finite collection S of distinct primes p, each satisfying $4p = t^2 + d$, for some integer t, such that the product of all the primes in S exceeds the bound M. By the discussion towards the end of Section 3.3, we can construct such a collection with high probability.
(b) For each p in S, consider a set of representatives for the $\overline{\mathbf{F}}_p$-isomorphism classes of elliptic curves over $\mathbf{F}_p$, and count the number of $\mathbf{F}_p$-points on each representative. In practice, we take as a representative the model

$$y^2 = x^3 + 3kx + 2k,$$

where $k = \frac{j}{1728-j}$, and j runs through all possible values in $\mathbf{F}_p$ (except 0 and 1728, which require special attention, and occur rarely). We then form the set S_p consisting of all the j-invariants such that the corresponding curve has $p+1+t$ or $p+1-t$ points. There are exactly h such j values, by Prop. 4.1 and Prop. 4.2 below (or by [Cox, p. 319]). Alternatively, for each representative, we could pick random points P on E and check if $(p+1)P = tP$ (or $(p+1)P = -tP$). This would rapidly filter out almost all of the candidates, and point-counting could be used to check the remaining ones.
(c) For each prime p in S, we form the polynomial $H_D(X) \bmod p$ by multiplying together the factors $(X - j)$, where j is in the set S_p. This is also justified by Prop. 4.1 and Prop. 4.2 below.

Version B: This can be used for any d and works unconditionally; however, we expect it to be more difficult to implement.

Version B is exactly like Version A except that we allow slightly more general primes when forming the set S in Step (a). We allow all primes p such that $4p = t^2 + u^2 d$, for some integers t and u. We again generate enough primes p so that their product exceeds the bound M; call the resulting set of primes T.

For each p in T, we compute the endomorphism ring for each $\overline{\mathbf{F}}_p$-isomorphism class of elliptic curves over $\mathbf{F}_p$ using the algorithm in [Kohel] (we use the same Weierstrass equations as in Step (b) of Version A above for the representatives of the isomorphism classes). We then form the set T_p consisting of all the j-invariants such that the corresponding curve has endomorhism ring isomorphic to $\mathcal{O}_K$. The class number of $\mathcal{O}_K$ is h, so there are exactly h such j values. If $u = 1$ for a prime p in T, then $T_p = S_p$, so we can just use Step (b) of Version A to compute T_p.

Finally, as in Step (c) of Version A, for each prime p in T, we form the polynomial $H_D(X) \bmod p$ by multiplying together the factors $(X - j)$, where j is in the set T_p.

Remark 3.1. Note that in Version B, when a primes p satisfies $4p = t^2 + u^2 d$, with $u > 1$, it is not sufficient to use point-counting to find the desired collection of elliptic curves (whose j-invariants form the set T_p). In that case, point-counting would produce the set of all elliptic curves E such that the endomorphism ring of E is an order in $\mathcal{O}_K$ that contains the order of index u (in $\mathcal{O}_K$). In this paper, we assumed that D was a fundamental discriminant, but to generalize our algorithm to general discriminants of orders in quadratic imaginary fields, it would be necessary to work with Version B of the algorithm. The number and size of the primes required to implement the two versions does not seem to be much different in practice (see the remark after Example 6.2). The main advantage to Version A is that it is easy to implement because there are many point-counting packages available. The main advantage to Version B is that it should generalize to work for all discriminants and is unconditional.

Step (2) Lift to $H_D(X) \bmod n$:

Consider the coefficients of $H_D(X)$ one at a time. Let i be an index for S (respectively for T) in Version A (respectively in Version B). For each i, let m_i be the corresponding element of S (respectively of T). For each coefficient, take x_i to be the corresponding coefficient in $H_D(X) \bmod m_i$ (which was computed in Step (1)) and use the modified CRT method of Section 5 to compute the corresponding coefficient in $H_D(X) \bmod n$. In other words, use the modified CRT method to use the knowledge of $H_D(X) \bmod p$ for p in S (respectively in T) to compute $H_D(X) \bmod n$. This step can be parallelized.

3.2. **Complexity analysis.** In our complexity analysis, we assume that if a and b are two integers, then their addition takes time $\mathrm{O}(\log a + \log b)$, their multiplication takes time $\mathrm{O}(\log a \log b)$, and the division of the greater by the smaller takes time $\mathrm{O}(\log a \log b)$. This can certainly be achieved by current algorithms; in fact, one can do better, but we will stick to our model of computation for the sake of simplicity and comparison (the complexity estimate for the Atkin-Morain algorithm given in [LL] does not assume fast arithmetic either). The steps mentioned below are numbered as in Section 3.1.

Step (0) According to [Cohen, §5.4], the computation of h can be done in time $\mathrm{O}(d^{1/4})$, or in time $\mathrm{O}(d^{1/5})$ assuming the generalized Riemann Hypothesis. The computation of B is dominated by the computation of $\sum 1/a$ in formula (2) which, takes negligible time (see [Cohen, §7.6.2], also [CraPom, p. 324]).

Step (1) We do the analysis only for Version A, since we do not know of a nice way of estimating the size or the number of the primes in the set T in Version B.
(a) By the discussion in §3.3, with high probability, the size of S is $\mathrm{O}(\frac{\log B}{\log d})$, and each $p \in S$ is $\mathrm{O}((\log B)^2)$; for the purposes of the complexity analysis, we will assume this happens (this makes our complexity analysis "probabilistic").
(b) The best implementations of elliptic curve point-counting algorithms currently run in time $\mathrm{O}((\log p)^5)$ (see [Schoof]), perhaps assuming fast arithmetic, although this will not affect the power of d in our overall complexity estimate. This step is repeated p times, so this step will take time $\mathrm{O}(p(\log p)^5)$. Finally, since the step is repeated for every prime in S, the total time taken will be $\mathrm{O}((\log B)^3(\log\log B)^5/\log d)$. In Section 3.3, we estimate $\log B$ in terms of d as $\log(B) = \mathrm{O}(\sqrt{d}(\log d)^2)$. Using this estimate, the time taken for this step in terms of d is $\mathrm{O}(d^{3/2}(\log d)^{10})$, ignoring $\log\log d$ factors. We should be able to speed up this step in practice by using the alternative suggested above to avoid counting points on each curve modulo p.
(c) The number of terms in the product used to compute $H_D(X) \bmod p$ is h and each coefficient is between zero and p, so this can be done in time $\mathrm{O}(h^2(\log p)^2)$, i.e., $\mathrm{O}(d(\log d)^2)$. Since the step has to be repeated for every $p \in S$, the total time taken is $\mathrm{O}(d^{3/2}(\log d)^3)$.

Overall, the total time taken by Step 1 in this version is $\mathrm{O}(d^{3/2}(\log d)^{10})$.

Step (2) As will be explained in Section 5, the time taken by the modified chinese remainder algorithm to compute all the coefficients of $H_D(X) \bmod n$ is

$$\mathrm{O}(d(\log d)^4 + d(\log d)^2 \log n + \sqrt{d}(\log n)^2).$$

Our algorithm differs from the one in [CNST, §4] mainly in Step (2). As shown in Section 5, if one uses the ordinary Chinese remainder theorem to find $H_D(X)$ and then reduces modulo n, as proposed in [CNST, §4], then the complexity of this procedure (i.e., the equivalent of Step (2) above) would be

$$\mathrm{O}(d(\log d)^2 \log n + d^{3/2}(\log d)^4),$$

which is not as good as our method in Step (2) when d is large (roughly speaking, bigger than $(\log n)^2$).

On the other hand, for primality proving as in [AtMor], one wants a small discriminant; in fact, in [LL, §5.10] they assume $d = \mathrm{O}((\log n)^2)$. In that case, it is clear that our algorithm is an improvement over the one in [CNST, §4] only if $\log B$ is bigger than $\log n$, i.e., if the coefficients of $H_D(X)$ are large compared to n.

The overall complexity of our algorithm, assuming Statement 3.1, is

$$\mathrm{O}(d^{3/2}(\log d)^{10} + d(\log d)^2 \log n + \sqrt{d}(\log n)^2).$$

3.3. **Some estimates needed for the complexity analysis.** We need an estimate for the size of B, i.e., an upper bound for the size of the coefficients of the Hilbert class polynomial $H_D(X)$. As follows from the discussion on p. 42 of [AtMor] (note that there is an apparent mistake in the formula for $\mathrm{Prec}(D)$ there), we may take

$$B = \binom{h}{\lfloor h/2 \rfloor} \exp\left(\pi\sqrt{d}\sum \frac{1}{a}\right), \tag{2}$$

where the sum in the expression above is taken over the set of integers a such that $ax^2 + bxy + cy^2$ is a primitive, reduced, positive definite binary quadratic form of discriminant D for some integers b and c (the set of such a's is finite). This bound is the product of all the roots times the largest binomial coefficient.

Note that by the corollary in [Lang1, Chap XVI, §4], we have $\log h \sim \log(\sqrt{d})$ as $d \to \infty$ (recall that the regulator of a quadratic imaginary field is one). This means that for any positive real number ϵ', we have $d^{1/2-\epsilon'} \leq h \leq d^{1/2+\epsilon'}$ when d is big enough. For the sake of simplicity in our analysis, we will assume $h \sim \sqrt{d}$.

We now give a lower bound on the size of $\log B$. By [Cohen, Lem. 5.3.4(1)], $a \leq \sqrt{d/3}$. Thus $\sum \frac{1}{a} \geq h\sqrt{\frac{3}{d}}$, and the latter is asymptotically a constant bigger than 1. Thus there is a constant $c > 1$ such that $\log B$ is greater than $c\sqrt{d}$ for d large enough.

To get an upper bound for $\log B$ in terms of d, we estimate $\sum \frac{1}{a}$ using the argument in [LL, p. 711]. They observe that there cannot be too many a's that are "small", since the the number of reduced forms (a, b) with a fixed a is bounded by $\tau(a)$, the number of positive divisors of a.

So certainly an overestimate for the sum $\sum \frac{1}{a}$ is given by $\sum_{a=1}^{d} \frac{\tau(a)}{a}$. This in turn can be written as a telescoping sum plus an error term:

$$\sum_{a=1}^{d} \frac{\tau(a)}{a} = \sum_{a=1}^{d} \left(\sum_{u=1}^{a} \tau(u)\right)\left(\frac{1}{a} - \frac{1}{a+1}\right) + \frac{1}{d+1}\sum_{a=1}^{d} \tau(a).$$

The sum $\sum_{a=1}^{d} \tau(a)$ can be estimated as $d \log d$ plus some lower order terms (see [NZM, Thm 8.28, p. 393]). So the first term can be estimated via the integral

$$\int_{a=1}^{d} \frac{\log a}{a} da = \frac{(\log d)^2}{2},$$

and the second term is less than $\log d$. This observation leads to the estimate

$$\sum \frac{1}{a} \leq \mathrm{O}(\log d)^2$$

(see also [CraPom, p. 324]). In fact, much better estimates for $\sum \frac{1}{a}$ should be possible, and it looks like a better bound is being assumed in the complexity analysis for the Atkin-Morain algorithm given by [LL], since they seem to assume that $\log(B) = \mathrm{O}(\sqrt{d})$, but we will stick with our estimate for our analysis.

Since the middle binomial coefficient is clearly less than the sum of all of the binomial coefficients, which is 2^h, we see that

$$B \leq 2^h e^{\pi\sqrt{d}(\log d)^2}.$$

So throughout the paper, we use the estimate

$$\log(B) = \mathrm{O}(\sqrt{d}(\log d)^2) = \mathrm{O}(h(\log h)^2). \tag{3}$$

An important consideration for accurately assessing the running time of our algorithm is the relative size of the small primes found in Step (1). Consider the following statement:

Statement 3.1. *If $d \not\equiv 7 \bmod 8$, then the procedure of finding primes in Version A of Step (1) terminates, and moreover, the size of the set S is $\mathrm{O}(\frac{\log B}{\log d})$ and each $p \in S$ is $\mathrm{O}((\log B)^2)$.*

We expect that the statement above is true with high probability when d is large enough. The main idea for Statement 3.1 was suggested to us by an anonymous referee. We now give a heuristic argument to support our expectation, some of the details of which were explained to us by J. Vaaler.

By the prime number theorem, the probability that a randomly chosen positive integer m is prime is $1/(\log m)$. For a given d, and randomly chosen t, we want to say that a number of the form $(t^2+d)/4$ looks like a randomly chosen integer, so that we can claim that the probability that it is prime is $1/\log((t^2+d)/4)$.

If $d \equiv 3 \mod 8$, say $d = 8k+3$, and if t is odd, say $t = 2\ell+1$, then $(t^2+d)/4 = \ell(\ell+1)+2k+1$ is an odd integer. If $d \equiv 4 \bmod 16$, say $d = 16k+4$, and if t is a multiple of 4, say $t = 4\ell$, then $(t^2+d)/4 = 4\ell^2+4k+1$ will be an odd integer. If $d \equiv 8 \bmod 16$ (the only possibility left), say $d = 16k+8$, and if t is even, say $t = 2\ell$, then $(t^2+d)/4 = \ell^2+4k+2$ will be an odd integer provided ℓ is odd. So for any d, for a random choice of an integer t, with probability at least $1/4$, the rational number $(t^2+d)/4$ will be an odd integer (i.e., $(t^2+d)/4$ will be an integer that need not necessarily be composite). So we will assume that the probability that it is prime (provided it is an odd integer) is indeed $1/\log((t^2+d)/4)$.

Now let c_1 and c_2 be two positive integers such that $c_1 < c_2$. Let S_1 denote the set

$$S_1 = \{(t^2+d)/4 : t \in \mathbf{Z}, c_1 \log B \le t \le c_2 \log B, (t^2+d)/4 \text{ is prime}\}.$$

The size of the set $\{(t^2+d)/4 : t \in \mathbf{Z}, c_1 \log B \le t \le c_2 \log B\}$ is $(c_2-c_1)\log B$, and roughly at least one-fourth of the elements of this set are integers. Moreover, among these latter integers, we are assuming that the probability that an element $(t^2+d)/4$ is prime is $1/\log((t^2+d)/4)$. Thus with high probability, the following statement should be true for large d:

($*$) The size of the set S_1 is between $\frac{1}{2}\left\lfloor \frac{(c_2-c_1)\log B}{4\log(c_2 \log B)} \right\rfloor$ and $2\left\lfloor \frac{(c_2-c_1)\log B}{4\log(d/4)} \right\rfloor$.

We will assume that ($*$) is indeed true for the rest of this section (so everything below holds only with high probability).

If $p \in S_1$, then

$$p \ge \frac{(c_1 \log B)^2 + d}{4} > \frac{(c_1 \log B)^2}{4}.$$

Thus

$$\sum_{p \in S_1} \log p > [2(\log\log B) + \log(c_1^2/4)] \frac{(c_2-c_1)\log B}{4\log(c_2 \log B)}.$$

By choosing c_1 and c_2 appropriately (say $c_2 = 12$ and $c_1 = 4$), we see that when d is large enough, $\sum_{p \in S_1} \log p > \log B$ and hence $\prod_{p \in S_1} p > B$.

Now let S_2 denote the set

$$S_2 = \{(t^2+d)/4 : t \in \mathbf{Z}, 0 \le t \le c_2 \log B, (t^2+4)/d \text{ is prime}\}.$$

Putting $c_1 = 0$ in statement ($*$), we see that the size of S_2 will be $\mathrm{O}(\frac{\log B}{\log d})$. Also, $\prod_{p \in S_1} p > B$, since the set S_2 contains the set S_1. Furthermore, if $p \in S_2$, then

$$p < ((c_2 \log B)^2 + d)/4.$$

Since d is $\mathrm{O}((\log B)^2)$, we see that p is $\mathrm{O}((\log B)^2)$. Finally (assuming statement ($*$) holds), the set S can be chosen to be a subset of the set S_2; from this, Statement 3.1 follows.

4. Computing $H_D(X)$ mod p for small primes p

In this section, we prove that Step 1 of our algorithm is a valid way to compute $H_D(X)$ mod p. The same strategy for this step was used in [CNST, §4], but it was not justified there, and the distinction between Versions A and B was blurred.

As in the introduction, let D be a fundamental discriminant and let $H_D(X)$ denote the associated Hilbert class polynomial. Let H denote the Hilbert class field of $K = \mathbf{Q}(\sqrt{D})$. Let p be a rational prime that splits completely in H, i.e., splits into principal ideals in K, which means that $4p = t^2 - Du^2$ for some integers u and t. Let $\mathcal{O}_K$ denote the ring of integers of K.

Let $\mathrm{Ell}(D)$ denote the set of isomorphism classes of elliptic curves over $\mathbf{C}$ with complex multiplication by $\mathcal{O}_K$ (i.e., whose ring of endomorphisms over $\mathbf{C}$ is isomorphic to $\mathcal{O}_K$). Then an equivalent way of defining the Hilbert class polynomial is as follows:

$$H_D(X) = \prod_{[E]\in \mathrm{Ell}(D)} (X - j(E)), \tag{4}$$

where, if E is an elliptic curve, then $j(E)$ denotes its j-invariant.

Let $\mathrm{Ell}'(D)$ denote the set of $\overline{\mathbf{F}}_p$-isomorphism classes of elliptic curves over $\mathbf{F}_p$ with endomorphism ring (over $\overline{\mathbf{F}}_p$) isomorphic to $\mathcal{O}_K$.

Proposition 4.1. *With notation as above,*

$$H_D(X) \bmod p = \prod_{[E']\in \mathrm{Ell}'(D)} (X - j(E')). \tag{5}$$

Proof. Let β be a prime ideal of the ring of integers of H lying over p. It follows from the discussion in the proof of Thm. 14.18 on p. 319-320 of [Cox] that in each class i in $\mathrm{Ell}(D)$, we can write down an elliptic curve E_i such that E_i is defined over H and E_i has good reduction modulo β (in fact, [Cox] gives a collection of such elliptic curves, denoted E_c; we just pick one such E_c for each class); denote the reduction modulo β of E_i by $\widetilde{E_i}$. Since p splits completely in H, $\widetilde{E_i}$ is defined over $\mathbf{F}_p$, as opposed to an extension of $\mathbf{F}_p$. Also, by [Lang, Chap 13, Thm. 12(ii)] (or [Cox, Thm. 14.16]), each $\widetilde{E_i}$ has endomorphism ring (over $\overline{\mathbf{F}}_p$) isomorphic to $\mathcal{O}_K$. This gives us a map ϕ from $\mathrm{Ell}(D)$ to $\mathrm{Ell}'(D)$. Since we assume that p splits in K, then by [Cox, Thm. 13.21], if two elliptic curves have distinct j-invariants, then the reductions modulo β of these j-invariants are distinct, i.e., the map ϕ is injective. By the Deuring lifting theorem [Lang, Chap. 13, Thm. 14] (or [Cox, Thm. 14.16]) this map is also a surjection.

From the definition of $j(E)$ in terms of the coefficients of the Weierstrass equation of E, it is easy to see that

$$H_D(X) \bmod p = \prod_{[E_i]\in \mathrm{Ell}(D)} (X - j(\widetilde{E_i})).$$

Hence, from the discussion above,

$$H_D(X) \bmod p = \prod_{[E']\in \mathrm{Ell}'(D)} (X - j(E')).$$

□

Proposition 4.2. *Suppose p is a prime and $x \neq 0$ is an integer such that $4p = x^2 - D$. Let E' be an elliptic curve over $\mathbf{F}_p$. Then $[E'] \in \mathrm{Ell}'(D)$ if and only if $\#E'(\mathbf{F}_p)$ is either $p+1-x$ or $p+1+x$.*

Proof. Suppose $\#E'(\mathbf{F}_p)$ is either $p+1-x$ or $p+1+x$. Let t denote the trace of the Frobenius endomorphism of E'. Then $t = x$ or $t = -x$. In either case, the discriminant of the characteristic polynomial $x^2 - tx + p$ of the Frobenius endomorphism is $t^2 - 4p = x^2 - 4p = D$. Let $\mathrm{End}(E')$ denote the endomorphism ring of E' and let R denote the subring of $\mathrm{End}(E')$ generated by the Frobenius endomorphism. Since $x \neq 0$, we have $\#E'(\mathbf{F}_p) \neq p+1$, hence by [Silv1, Ex. 5.10b], E' is not supersingular, i.e., $\mathrm{End}(E')$ as an order in a quadratic imaginary field. The quotient field of R is K, and so $\mathrm{End}(E')$ as an order in K. Since D is a fundamental discriminant, $R = \mathcal{O}_K$. At the same time $\mathrm{End}(E')$ is contained in $\mathcal{O}_K$ (as $\mathcal{O}_K$ contains all orders of K). Hence $\mathrm{End}(E') = \mathcal{O}_K$, i.e., $[E'] \in \mathrm{Ell}'(D)$.

Conversely, suppose $[E'] \in \mathrm{Ell}'(D)$ and let t denote the trace of the Frobenius endomorphism of E'. Suppose the Frobenius endomorphism generates a subring of index u in the endomorphism ring of E' (which is $\mathcal{O}_K$). Then the characteristic polynomial $x^2 - tx + p$ of the Frobenius endomorphism has discriminant $u^2 D$, hence $4p = t^2 - u^2 D$. But we know $4p = x^2 - D$, so by [Cox, Ex. 14.17], $t = x$ or $t = -x$. Hence $\#E'(\mathbf{F}_p)$ is either $p + 1 - x$ or $p + 1 + x$. □

5. A modification of the Chinese remainder theorem

5.1. **The algorithm and its complexity.** This section follows [Couv, §2.1] closely, which in turn is based on [MS, §4]; the only addition is a more detailed complexity analysis.

The problem we consider is as follows: for some positive integer ℓ we are given a collection of pairwise coprime positive integers m_i for $i = 1, 2, \ldots, \ell$. For each i, we are also given an integer x_i with $0 \le x_i < m_i$. In addition, we are given a small positive real number ϵ. Finally, we are told that there is an integer x such that $|x| < (1/2 - \epsilon)\prod_i m_i$ and $x \equiv x_i \bmod m_i$ for each i; clearly such an integer x is unique if it exists. The question is to compute $x \bmod n$, for a given positive integer n.

Define

$$M = \prod_i m_i \tag{6}$$

$$M_i = \prod_{j \ne i} m_j = M/m_i \tag{7}$$

$$a_i = 1/M_i \bmod m_i, \qquad 0 \le a_i < m_i. \tag{8}$$

Then the number $z = \sum_i a_i M_i x_i$ is congruent to x modulo M. Hence, if $r = \left\lfloor \frac{z}{M} + \frac{1}{2} \right\rfloor$, then $x = z - rM$. So $x \bmod n = z \bmod n - (r \bmod n)(M \bmod n)$; the point is that we can calculate $r \bmod n$ without calculating z, as we now explain. From the fact that $x = z - rM$ and $|x| < (1/2 - \epsilon)M$, it follows that $\frac{z}{M} + \frac{1}{2}$ is not within ϵ of an integer. Hence, to calculate r, one only has find an approximation t to z/M such that $|t - z/M| < \epsilon$, and then round t to the nearest integer. Such an approximation t can be obtained from

$$\frac{z}{M} = \sum_i \frac{a_i x_i}{m_i}, \tag{9}$$

where the calculations are done using floating point numbers.

If a and b are two integers, then let $\mathrm{rem}(a, b)$ denote the remainder of the Euclidean division of a by b; we will assume that it takes time $\mathrm{O}(\log a \log b)$ to calculate $\mathrm{rem}(a, b)$ and $\gcd(a, b)$.

From the discussion above, we obtain the following algorithm:
(i) Compute a_i's, for each i, using (8): this takes time $\mathrm{O}(\sum_i(\sum_j(\log m_j \log m_i) + \ell(\log m_i)^2 + (\log m_i)^2)) = \mathrm{O}((\log M)^2 + \ell \sum(\log m_i)^2)$.
(ii) Compute $\mathrm{rem}(M, n)$ using (6): this will take time $\mathrm{O}(\sum_i(\log m_i \log n) + \ell(\log n)^2) = \mathrm{O}(\log n \log M + \ell(\log n)^2)$.
(iii) Compute $\mathrm{rem}(M_i, n)$ for each i by dividing $\mathrm{rem}(M, n)$ by m_i modulo n: this will take time $\mathrm{O}(\ell(\log n)^2)$ (in our application, m_i will be much lesser than n), and can be parallelized.
(iv) Compute r: In (9), every term in the sum has to be calculated to precision ϵ/ℓ, hence the calculation of each term takes time $\mathrm{O}((\log(\ell/\epsilon))^2)$. In the application to computing $H_D(X) \bmod n$, we take ϵ to be an arbitrary small number and take $M = B/(1/2 - \epsilon)$. Then the calculation of all the terms in (9) will take total time $\mathrm{O}(\ell(\log \ell)^2)$ and the addition in (9) of ℓ numbers with precision ϵ/ℓ will take time $\mathrm{O}(\ell \log \ell)$.
(v) Output $\mathrm{rem}(x, n) =$

$$\mathrm{rem}\Bigg(\bigg(\mathrm{rem}\Big(\sum_i(\mathrm{rem}(a_i \cdot x_i, n) \cdot \mathrm{rem}(M_i, n)), n\Big) - \mathrm{rem}(r, n) \cdot \mathrm{rem}(M, n)\bigg), n\Bigg). \tag{10}$$

The various substeps in step (v) and the time taken for each are as follows:
(a) Calculation of $\mathrm{rem}(a_i \cdot x_i, n)$ and $\mathrm{rem}(M_i, n)$ for all i: takes time $\mathrm{O}(\sum_i((\log m_i)^2 + (\log m_i)(\log n)))$.
(b) Computing the product of $\mathrm{rem}(a_i \cdot x_i, n)$ and $\mathrm{rem}(M_i, n)$ for all i: takes time $\mathrm{O}(\ell(\log n)^2)$.
(c) Performing the sum in (10) and taking remainder modulo n: this involves about ℓ additions of integers of size up to ℓn^2, which takes time $\mathrm{O}(\ell \log(\ell n^2))$ and taking the remainder takes time $\mathrm{O}((\log n)(\log(\ell n^2))$.
(d) Calculation of $\mathrm{rem}(r, n) \cdot \mathrm{rem}(M, n)$: The size of r is about $\sum m_i$, hence this substep takes time $\mathrm{O}((\log n)\log(\sum m_i) + (\log n)^2)$.
(e) Subtraction operation and taking remainder: takes time $\mathrm{O}(\log n)$ and $\mathrm{O}((\log n)^2)$ respectively.

In Section 3, we use this algorithm compute each coefficient in $H_D(X) \bmod n$ from the knowledge of the corresponding coefficient in $H_D(X) \bmod p$ for all $p \in S$. Then steps (i), (ii), and (iii) above are common to the lifting of all the coefficients, and only step (iv) and (v) have to be repeated for each coefficient.

In the notation of Section 3, the m_i's are the elements of S, and so, assuming Statement 3.1, we see that m_i's are $\mathrm{O}((\log B)^2)$ and ℓ is $\mathrm{O}(\log B/\log d)$. Using this, and the estimates from §3.3, we see that the most time consuming steps are Step (i), which takes time $\mathrm{O}(d(\log d)^4)$, and Steps (v-a) and (v-d) repeated h times, which take time $\mathrm{O}(d(\log d)^2 \log n)$ and $\mathrm{O}(\sqrt{d}(\log n)^2)$ respectively.

5.2. **Complexity of the usual Chinese Remainder Algorithm.** If we are to use the naive Chinese remainder theorem for the problem stated at the beginning of Section 5.1, then we calculate

$$z = \mathrm{rem}\Big(\Big(\sum_i a_i \cdot x_i \cdot M_i\Big), M\Big), \tag{11}$$

and then reduce z modulo n.

The steps involved are as follows:
(i) Compute a_i's, for each i, using (8): this takes time $\mathrm{O}(\sum_i(\sum_j(\log m_j \log m_i) + \ell(\log m_i)^2 + (\log m_i)^2)) = \mathrm{O}((\log M)^2 + \ell\sum(\log m_i)^2)$.
(ii) Calculation of $a_i \cdot x_i \cdot M_i$ for all i: takes time $\mathrm{O}(\sum_i(\log m_i)(\log M))$.
(iii) Performing the sum in (11): this involves ℓ additions of integers of size up to $\ell m_i^2 M$, hence takes time $\mathrm{O}(\ell\log(\ell m_i^2 M))$.
(iv) Calculating the outer "rem" in (11): takes time $\mathrm{O}((\log M)\log(\ell m_i^2 M))$.
(v) Reducing z modulo n: takes time $\mathrm{O}((\log M)(\log n))$.

In the context of lifting $H_D(X) \bmod p$ to $H_D(X)$ and then reducing $H_D(X)$ modulo n, only steps (ii) – (vi) have to be repeated for each coefficient. In the notation of Section 3, the m_i's are the elements of S, and so, assuming Statement 3.1, we again have that m_i's are $\mathrm{O}((\log B)^2)$ and ℓ is $\mathrm{O}(\log B/\log d)$. Using this, and the estimates from §3.3, we see that the most time consuming steps are Steps (ii) and (iv), each of which take total time $\mathrm{O}(d^{3/2}(\log d)^4)$ and Step (v), which takes total time $\mathrm{O}(d(\log d)^2 \log n)$.

From this analysis, we see that our modified Chinese remainder algorithm will be asymptotically more efficient than the usual one when $\log B$ is large compared to $\log n$, which will certainly be the case in our context whenever $d \geq (\log n)^2$ (certainly, when $n > B$, the modified version is no better than the usual Chinese remainder algorithm).

6. Examples

In this section we present several examples to illustrate our algorithm. Throughout these examples, we used the software package PARI, which is available at

`http://www.parigp-home.de`

6.1. $D = -59$.

6.1.1. *Atkin-Morain Method.* Since here we are dealing with a very small discriminant, we can easily compute the minimal polynomial over the integers directly by finding all the reduced, positive definite, primitive, binary quadratic forms with discriminant -59 and then evaluating $j(\tau)$ for the corresponding τ with sufficiently high precision. The class number of $\mathbf{Q}(\sqrt{-59})$ is three, and the three binary quadratic forms are

$$(a, b, c) = (3, 1, 5), (3, -1, 5), (1, 1, 15).$$

The corresponding algebraic integer is

$$\tau_{(a,b,c)} = \frac{-b + \sqrt{b^2 - 4ac}}{2a}.$$

We expect the absolute value of the largest of the $j(\tau)$ to be roughly $e^{\pi\sqrt{59}} \approx e^{24}$. Evaluating the product

$$(x - j(\tau_1))(x - j(\tau_2))(x - j(\tau_3))$$

with enough significant digits and rounding the coefficients to integers, we find the class polynomial:

$$H_D(x) = x^3 + 30197678080x^2 - 140811576541184x + 374643194001883136.$$

Here 28 decimal digits of precision are required using the package pari (19 digits of precision are not enough). Note that e^{63} translates to about 28 digits and the estimate for the amount of precision required given in [BSS] [Cohen] [AtMor] [CNST] is about 63.

6.1.2. *Chinese Remainder type algorithms.* To implement our algorithm for this example, we set the bound B equal to e^{41} to be bigger than the largest coefficient of $H_D(x)$. This estimate comes from the product of the three j values, whose absolute value we expect to be roughly

$$e^{\pi\sqrt{59}(1+\frac{1}{3}+\frac{1}{3})}.$$

We find the following list of 7 small primes which are of the form $(t^2 - D)/4$ for some integer t:

$$17, 71, 197, 521, 827, 1907, 3797, 5417$$

and whose product exceeds B. For each prime p in the list, we loop through the $p - 1$ possible j-values. For each possible j-value, we count the number of points on a curve over $\mathbf{F}_p$ with that j-value using a version of Schoof's algorithm (we use a version available on the web by Mike Scott: `ftp://ftp.compapp.dcu.ie/pub/crypto/sea.cpp`). If the curve has either $p+1+t$ or $p+1-t$ points, with $t^2 = 4p - 59$, then we keep that j-value in a list S_p. At the end of the loop, we will have h j-values in the list S_p, where h is the degree of $H_D(x)$. Then the polynomial $H_D(x) \mod p$ is formed as the product over $j \in S_p$ of $(x - j)$.

Here is a table summarizing the results for this example:

TABLE 1

p	t	$j \in S$	$H_D(x) \mod p$
17	3	$j = 2, 7, 13$	$x^3 + 12x^2 + 12x + 5$
71	15	$j = 51, 54, 67$	$x^3 + 41x^2 + 62x + 11$
197	27	$j = 71, 195, 130$	$x^3 + 195x^2 + 160x + 139$
521	45	$j = 103, 366, 367$	$x^3 + 206x^2 + 379x + 510$
827	57	$j = 97, 498, 554$	$x^3 + 505x^2 + 824x + 196$
1907	87	$j = 24, 915, 1613$	$x^3 + 1262x^2 + 1432x + 1045$
3797	123	$j = 70, 958, 2381$	$x^3 + 388x^2 + 1114x + 1584$

Usual Chinese Remainder routine

Here is a short routine in the algebraic number theory package PARI to compute the polynomial $H_D(x)$ with integer coefficients using the usual Chinese Remainder Theorem. It takes as input the coefficients of $H_D(x)$ modulo the small primes p.

```
l=7; (number of small primes)
h=degree; (degree of the hilbert class polynomial)
m=[17,71,197,521,827,1907,3797]; (list of small primes)
M=prod(i=1,l,m[i]); (M=17*71*197*521*827*1907*3797)
log(M);
invm = vector(l,i,M/m[i]);
a=vector(l,i,Mod(1/invm[i],m[i]));
modcoeff = [[12,41,195,206,505,1262,388],[12,62,160,379,824,1432,1114],
[5,11,139,510,196,1045,1584]]; (list of coefficients modulo small primes)
z=vector(h,j,Mod(sum(i=1,l,lift(a[i])*invm[i]*modcoeff[j][i]),M));
```

Modified Chinese Remainder routine

For our algorithm, we input in addition the prime n such that we want to determine $H_D(x) \mod n$. Here is a short routine in PARI to compute the polynomial $H_D(x)$ with coefficients modulo n using our modified version of the Chinese Remainder Theorem.

```
n=prime; (the prime where we want the curve in the end)
r=vector(h,j,round(sum(i=1,l,(lift(a[i])*modcoeff[j][i]/m[i])))))
finalcoeff=vector(h,j,sum(i=1,l,
        lift(a[i])*modcoeff[j][i]*Mod(invm[i],n))-Mod(r[j],n)*Mod(M,n))
```

Note that the precision required for this computation is almost trivial (the minimum value to set the precision in PARI is 9 significant digits).

n=141767

Here is an example where we use our algorithm to find the class polynomial modulo n. Note that $4n = 753^2 - D$, so we will construct a curve over $\mathbf{F}_n$ with $142521 = n + 1 + 753$ points. The output of our Modified Chinese Remainder routine is:

$$[\mathrm{Mod}(31177, 141767), \mathrm{Mod}(73152, 141767), \mathrm{Mod}(48400, 141767)].$$

Note that this corresponds to the class polynomial that we found using the Atkin-Morain method reduced modulo n:

$$X^3 + 31177X^2 + 73152X + 48400.$$

Taking the root $j = 118481 \bmod n$, we get the elliptic curve

$$y^2 = x^3 + 39103x + 120580.$$

It has 142521 points as desired.

Remark 6.1. Actually, the third coefficient in this example had to be re-computed because there was a rounding problem. The constant term of the class polynomial over the integers is slightly more than half the product of the small primes. The problem in this example can be solved in a clean way by adding one more prime to the algorithm, since in fact our algorithm requires the product of the small primes to slightly exceed $2B$ by an amount depending on the choice of epsilon: $B/(1/2 - \epsilon)$.

6.2. $D = -832603$**.** The Algorithms and Parameters for Secure Electronic Signatures document put out by the EESSI-SG (European Electronic Signature Standardisation Initiative Steering Group) recommends using elliptic curves with class number of the endomorphism ring at least equal to 200. Here is an example with class number equal to 96. Let

$$n = 100959557.$$

Note that $4n = 20075^2 - D$, and so we will construct a curve over $\mathbf{F}_n$ with $N = 100979633 = n + 1 + 20075$ points.

We have that D is square-free and $\mathbf{Q}(\sqrt{D})$ has class number $h = 96$, which is small compared to the square root of $|D|$,

$$\sqrt{|D|} \approx 912.$$

According to the estimates, the largest coefficient of the class polynomial is bounded by e^{5368}. This comes from the fact that

$$\sum_{i=1}^{96} \frac{1}{a} \approx 1.85$$

and the middle binomial coefficient is roughly e^{64}.

6.2.1. *Atkin-Morain method.* We can obtain the class polynomial using the algorithm of Atkin and Morain with 3000 digits of precision (2332 digits should suffice):

$x^{96}+$ 8986950689916460612768050899826095370126160959774067006607495722714787327536405
11195940426329493962250363608918506814518954357512108376324309765509813261009595
72615396095460780845684222178125741276615369508754546593823796954336290438786044
54401141760139087536761069637319825798963352735300683356996983745448647386647867
26063390570575243929901907691553896874023573783820406248132762600812249711372047
55882861009465622598491768081847062179144325418471334571707958715942702715493314
88237482404374709398037956562818329691426448791666838198286258706039015068098946
24510449977402232596376577346850486922319170621800447828468015555155662062177391
08385797919357857853408831828384120143178961174846657318648760182117564137653818
31734687436523641791869559811287475164662560558340565130954332294988968060971888
71815593515878469206432064483048317524773444570122581660831541350800516869161291
01483247617224314616156733489349276043330450686852025326165481636562782630791850
43524347061886083145402858558832786452505054211954992588893518489408407045712834
80364209087452918765509915544167886763955395126621398677472529126929317764001654
07674073078383580568650075515962375620983618886988248866522341997936320370535130
54749569703659745187123040222118255096012800000x^{95} ...

The coefficients of this polynomial are so big that it would take about 30 pages just to write the polynomial down in that font size.

6.2.2. *Modified Chinese Remainder Algorithm.* To obtain the class polynomial using our method, we first make a list of primes p which are each of the form $4p = t^2 - D$ for some integer t.

List of small primes

[208207, 208223, 208261, 208283, 208333, 208391, 208457, 208493, 208657, 208907, 208963, 209021, 210131, 210407, 210601, 210803, 210907, 211231, 211457, 211573, 211691, 211811, 211933, 212057, 212183, 212573, 212843, 212981, 213263, 213407, 213553, 214003, 214631, 215123, 215461, 215983, 216523, 217081, 217271, 217463, 218453, 218657, 219071, 219281, 219707, 220141, 220361, 220807, 221261, 221723, 221957, 222193, 222913, 223403, 224153, 224921, 226241, 226511, 226783, 227611, 228457, 229321, 229613, 230203, 230501, 232643, 233591, 233911, 235211, 235541, 236207, 236881, 237563, 240371, 241093, 241823, 245593, 245981, 246371, 247553, 248351, 248753, 249563, 249971, 250793, 251623, 253307, 253733, 254161, 255023, 255457, 256771, 257657, 258551, 259001, 259453, 259907, 260363, 261281, 263611, 264083, 265511, 266957, 268913, 273943, 274457, 274973, 275491, 276011, 278111, 279173, 279707, 280243, 281321, 282407, 283501, 284051, 286831, 287393, 290233, 292541, 293123, 294293, 298451, 299053, 303323, 304561, 307691, 308323, 310231, 315407, 320041, 321383, 322057, 324773, 326143, 326831, 328213, 329603, 333821, 335957, 339557, 340283, 343943, 344681, 348401, 349903, 350657, 351413, 352931, 356761, 364571, 366953, 367751, 368551, 369353, 373393, 375841, 379133, 382457, 387503, 388351, 397811, 398683, 399557, 401311, 403957, 404843, 405731, 411101, 414721, 415631, 416543, 417457, 418373, 419291, 421133, 422057, 424841, 426707, 429521, 436157, 437113, 441923, 444833, 445807, 448741, 450707, 453671, 456653, 457651, 458651, 460657, 466723, 468761, 474923, 475957, 480113, 481157, 482203, 483251, 484301, 486407, 487463, 490643, 491707, 495983, 499211, 503543, 504631, 507907, 510101, 511201, 513407, 514513, 515621, 520073, 523433, 527941, 531343, 538201, 539351, 541657, 543971, 545131, 548623, 550961, 555661, 556841, 560393, 568751, 571157, 573571, 579641, 582083, 593171, 598151, 606943, 608207, 610741, 612011, 615833, 620957, 622243, 623531, 626113, 637831, 639143, 640457, 644411, 647057, 648383, 652373, 653707, 655043, 659063, 665803, 668513, 672593, 678061, 679433, 682183, 687707, 689093, 691871, 696053, 707293, 712961, 718661, 720091, 724393, 727271, 728713, 730157, 731603, 735953, 738863, 740321, 741781, 744707, 759457, 763921, 766907, 771401, 772903, 777421, 783473, 788033, 789557, 794141, 797207, 800281, 806453, 814213, 817331, 823591, 825161, 829883, 831461, 834623, 839381, 844157, 845753, 853763, 861823, 865061, 866683, 874823, 883013, 897881, 901207, 902873, 906211, 911233, 912911, 914591, 916273, 921331, 924713, 934907, 940031, 950333, 952057, 955511, 957241, 958973, 967663, 969407, 971153, 972901, 974651, 976403, 978157, 983431, 997583, 1006493, 1015453, 1020853, 1028081, 1031707, 1035341, 1040807, 1042633, 1048123, 1055471, 1066553, 1068407, 1075843, 1077707, 1081441, 1083311, 1088933, 1094573, 1107803, 1115407, 1119221, 1124957, 1130711, 1132633, 1134557, 1136483, 1138411, 1140341, 1146143, 1151963, 1161703, 1167571, 1187261, 1195193, 1197181, 1201163, 1209151, 1217171, 1221193, 1223207, 1225223, 1227241, 1231283, 1241423, 1278341, 1286633, 1288711, 1290791, 1294957, 1301221, 1309601, 1311701, 1315907, 1318013, 1328573, 1330691, 1334933, 1337057, 1347707, 1351981, 1362701, 1377793, 1379957, 1382123, 1388633, 1392983, 1403893, 1406081, 1410463, 1417051, 1419251, 1425863, 1430281, 1432493, 1434707]

The list contains 410 primes. Their product is roughly e^{5379}, which exceeds the bound $2B$ as desired.

To illustrate the algorithm, we find the class polynomial modulo the largest prime on the list $p = 1434707$. Note that $4p = 2215^2 - D$. By counting the number of points on a representative for each isomorphism class of elliptic curves over $\mathbf{F}_p$, we found the following list of 96 j-values such that the associated elliptic curve has $p + 1 \pm 2215$ points over $\mathbf{F}_p$.

j-values for $p = 1434707$:

[28534, 29664, 39989, 50559, 58497, 61669, 87155, 97333, 120663, 153566, 158121, 164378, 182440, 199741, 210115, 218108, 219599, 237389, 257474, 289215, 317239, 333891, 335757, 365925, 381504, 395862, 403801, 449952, 482780, 485134, 487074, 511916, 527120, 543027, 574978, 583669, 584091, 585813, 595906, 642664, 644346, 653188, 654512, 655573, 696063, 698345, 699985, 702445, 705943, 710770, 721309, 738498, 759603, 780978, 795085, 816076, 821241, 869331, 871700, 889175, 897281, 902226, 923156, 924382, 980018, 1022428, 1033432, 1057121, 1079631, 1093031, 1101285, 1129437, 1154957, 1161878, 1175298, 1185913, 1186864, 1199076, 1205398, 1231078, 1252451, 1279055, 1281872, 1286184, 1312922, 1327236, 1334297, 1352254, 1352769, 1364919, 1368722, 1381024, 1410659, 1426507, 1428519, 1431597]

We find that

$$H_D(X) \bmod p =$$

$$X^{96}+1163995X^{95}+922656X^{94}+700837X^{93}+1079920X^{92}+466732X^{91}+154378X^{90}+399013X^{89}+744868X^{88}+1140439X^{87}+238431X^{86}+439229X^{85}+1168335X^{84}+1088371X^{83}+1065323X^{82}+923089X^{81}+370237X^{80}+418673X^{79}+26462X^{78}+1186790X^{77}+577727X^{76}+1026750X^{75}+1311499X^{74}+42221X^{73}+1226509X^{72}+1302356X^{71}+1205738X^{70}+706055X^{69}+916474X^{68}+870490X^{67}+940463X^{66}+779702X^{65}+543453X^{64}+1023692X^{63}+985646X^{62}+734246X^{61}+744646X^{60}+754597X^{59}+67621X^{58}+394070X^{57}+801259X^{56}+1203063X^{55}+1415480X^{54}+182257X^{53}+358715X^{52}+659376X^{51}+343711X^{50}+472997X^{49}+545620X^{48}+578548X^{47}+223638X^{46}+281011X^{45}+170375X^{44}+514817X^{43}+327182X^{42}+506290X^{41}+550176X^{40}+157534X^{39}+1257296X^{38}+1245604X^{37}+311058X^{36}+532467X^{35}+601208X^{34}+1069781X^{33}+52757X^{32}+508590X^{31}+247205X^{30}+1293507X^{29}+1089763X^{28}+326605X^{27}+46947X^{26}+1147567X^{25}+884035X^{24}+535907X^{23}+1164336X^{22}+952400X^{21}+1245681X^{20}+348341X^{19}+43230X^{18}+1201679X^{17}+486702X^{16}+360056X^{15}+28756X^{14}+1068784X^{13}+993753X^{12}+790102X^{11}+436946X^{10}+37636X^{9}+459204X^{8}+1185717X^{7}+644728X^{6}+1031301X^{5}+384651X^{4}+380850X^{3}+1358865X^{2}+1127134X+401105 \bmod p.$$

The class polynomials modulo the other 409 primes are not included here. This polynomial indeed corresponds to the reduction modulo p of the class polynomial found using the Atkin-Morain algorithm.

Remark 6.2. In this example, we find that allowing primes p such that $4p = u^2 + v^2 d$ (i.e., using Version B) does not help much. The size of v is constrained by the desire to keep the primes small; here, v must satisfy $v \leq 2$ to avoid getting larger primes. Allowing $v = 2$, we still need a list of length 410 primes to exceed the bound. The black art of balancing the size of the primes with the number of primes required is not the subject of this paper, but at least in this example Version B seems no better in this regard.

References

[AtMor] Atkin, A. O. L., Morain, F., *Elliptic curves and primality proving*, Math. Comp. 61 (1993), no. 203, 29–68.

[BSS] Blake, I., Seroussi, G., Smart, N., *Elliptic curves in cryptography*, LMS Lecture Note Series, 265, Cambridge University Press, Cambridge, 2000.

[CNST] Chao, J., Nakamura, O., Sobataka, K., Tsujii, S., *Construction of secure elliptic cryptosystems using CM tests and liftings*, Advances in Cryptology, ASIACRYPT'98(Beijing), 95–109, Lecture Notes in Comput. Sci., 1514, Springer, Berlin, 1998.

[Cohen] Cohen, H., *A course in computational number theory*, GTM 138, Springer-Verlag, Berlin, 1993.

[Couv] Couveignes, J.-M., *Computing a square root for the number field sieve*, The development of the number field sieve, 95–102, Lecture Notes in Math., 1554, Springer, Berlin, 1993.

[Cox] Cox, D. A., *Primes of the form $x^2 + ny^2$*: Fermat, class field theory and complex multiplication, John Wiley & Sons, Inc., New York, 1989.

[CraPom] Crandall, R.; Pomerance, C., *Prime numbers. A computational perspective*, Springer-Verlag, New York, 2001.

[Kohel] Kohel, D., *Endomorphism rings of elliptic curves over finite fields*, Ph. D. thesis, University of California, Berkeley.
`http://www.maths.usyd.edu.au/u/kohel/`

[Lang1] Lang, S., *Algebraic Number Theory*, Second Edition, Graduate Texts in Mathematics, 110. Springer-Verlag, New York-Berlin, 1994.

[Lang] Lang, S., *Elliptic functions*, Second edition, Graduate Texts in Mathematics, 112. Springer-Verlag, New York-Berlin, 1987.

[LL] Lenstra, A. K.; Lenstra, H. W., Jr., *Algorithms in number theory*, Handbook of theoretical computer science, Vol. A, 673–715, Elsevier, Amsterdam, 1990.

[MS] Montgomery, P. L.; Silverman, R. D, *An FFT extension to the $P-1$ factoring algorithm*, Math. Comp. 54 (1990), no. 190, 839–854.

[NZM] Niven, Ivan; Zuckerman, Herbert S.; Montgomery, Hugh L. *An introduction to the theory of numbers*, Fifth edition. John Wiley & Sons, Inc., New York, 1991.

[Schoof] Schoof, R., *Counting points on elliptic curves over finite fields*, J. Théor. Nombres Bordeaux 7 (1995), no. 1, 219–254.

[Silv1] Silverman, J., *The Arithmetic of Elliptic Curves*, Graduate Texts in Mathematics, 106, Springer-Verlag, New York, 1986.

[Silv2] Silverman, J., *Advanced topics in the arithmetic of elliptic curves*, Graduate Texts in Mathematics, 151, Springer-Verlag, New York, 1994.

Department of Mathematics, University of Texas, Austin, Texas, USA
E-mail address: amod@math.utexas.edu

Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA.
E-mail address: klauter@microsoft.com

Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA.
E-mail address: venkie@microsoft.com