Constructing elliptic curves with a given number of points over a finite field

Amod Agashe, Kristin Lauter, Ramarathnam Venkatesan

November 13, 2001

Abstract. In using elliptic curves for cryptography, one often needs to construct elliptic curves with a given or known number of points over a given finite field. 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 the Hilbert class polynomial $H_D(X)$ modulo some integer p for a certain fundamental discriminant D. The usual way of doing this is to compute $H_D(X)$ over the integers and then reduce modulo p. But this involves computing the roots with very high accuracy and subsequent rounding of the coefficients to the closest integer. (Such accuracy issues also arise for higher genus cases.) We present a modified version of the Chinese remainder theorem (CRT) to compute $H_D(X)$ modulo p directly from the knowledge of $H_D(X)$ modulo enough small primes. Our algorithm is inspired by Couveigne's method for computing square roots in the number field sieve, which is useful in other scenarios as well. It runs in heuristic expected time less than the CRT method in [CNST]. Moreover, our method requires very few digits of precision to succeed, and avoids calculating the exponentially large coefficients of the Hilbert class polynomial over the integers.

1 Introduction

In the use of elliptic curves in cryptography, one often needs to construct elliptic curves with a given (or known) number of points over a given finite field, $\mathbf{F}_p$. One way of doing this is to randomly pick elliptic curves and then to count the number of points on the curve over the finite field, repeating this until the desired order is found. Atkin pointed out that instead, one can use the theory of complex multiplication to construct elliptic curves with known number of points. Although asymptotically 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 an improvement to the complex multiplication method, decreasing both the running time and the amount of precision required for the algorithm to succeed.

One of the steps in the complex multiplication method is the calculation of the Hilbert class polynomial $H_D(X)$ modulo some integer p for a certain fundamental discriminant D. The usual way of doing this is to compute $H_D(X)$ over the integers and then reduce modulo p. Atkin and Morain proposed to compute $H_D(X)$ as an integral polynomial by listing all the relevant binary quadratic forms, evaluating the j-function as a floating point integer with sufficient precision, and then taking the product and rounding the coefficients to nearest integers.

In [CNST, §4], the authors suggest computing $H_D(X) \bmod p_i$ for sufficiently many small primes p_i and then using the Chinese remainder theorem to compute $H_D(X)$ as a polynomial with integer coefficients. In this paper we use a modified version of the Chinese remainder theorem to compute $H_D(X)$ modulo p directly (knowing $H_D(X) \bmod p_i$ for sufficiently many small primes p_i). By avoiding computing the coefficients of $H_D(X)$ as integers, we obtain an algorithm with shorter running time and requiring less precision of computation. Whereas for large discriminants the Atkin-Morain algorithm becomes impractical because the size of the coefficients as integers is approximately $e^{\pi d}$, our algorithm only computes the coefficients modulo p, requiring less space to write down and not requiring $e^{\pi d}$ digits of precision to compute. An analysis of the improvement we obtain in the running time is provided in Section 5. It is hoped that our methods and analysis will help in adapting Couveignes style solutions to other problems, say in higher genus cases.

In designing cryptosystems, restricting to small discriminants may leave the cryptosystem vulnerable to some yet unknown attacks; hence it is desirable to consider large discriminants. On the other hand, for primality proving as in [AtMor], small discriminants are sufficient; in fact, they assume $d = \mathrm{O}((\log p)^2)$ (see [LL, §5.10]). In our analysis, we often make the same assumption on the relative size of d and p, even though in practice, d could be much larger. The relative size of d depends on how close N is to the end of the Weil interval (since $D = t^2 - 4p$). If p has n bits and we choose $t = [2\sqrt{p}]$, then d could be small, depending on how close $4p$ is to a perfect square. However, even if we take t to be one less, the size of d could easily be something like $\sqrt{p}$ with size $n/2$ bits.

In Section 2, we give a brief description of the CM 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 the steps of the algorithm. Finally in Section 7, we give some examples of our method.

2 Complex multiplication method

We briefly review the complex multiplication method, referring the reader to [AtMor] and [Silv2] for details. Suppose we are given a prime p, and a non-negative number N in the Hasse-Weil interval $[p+1-2\sqrt{p}, p+1+2\sqrt{p}]$. We want to produce an elliptic curve E over $\mathbf{F}_p$ with N $\mathbf{F}_p$-points:

$$\#E(\mathbf{F}_p) = N = p + 1 - t,$$

where t will be the trace of the Frobenius endomorphism of E over $\mathbf{F}_p$ We set

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

The Frobenius endomorphism of E has characteristic polynomial

$$x^2 - tx + p.$$

It follows from the quadratic formula that the roots of this polynomial lie in $\mathbf{Q}(\sqrt{D})$. It is standard to associate the Frobenius endomorphism with a root of this polynomial. If E is not supersingular, then R, the endomorphism ring of E, is an order in the ring of integers of $K = \mathbf{Q}(\sqrt{D})$ ([Silv1]). For simplicity of the algorithm, we will want to assume that R is the full ring of integers. If D is square-free then this condition is automatically satisfied, since then the Frobenius generates the full ring of integers and is contained in the endomorphism ring. These results can be generalized to orders in the ring of integers, but the algorithm will become more complicated. 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 $D \equiv 8, 12 \bmod 16$. We will assume for simplicity that D is a fundamental discriminant.

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 R. Asymptotically, h is approximately $\sqrt{d}$, $d = |D|$. It is known that $H_D(X)$ has integer coefficients, and that it generates H, the Hilbert class field of K. (If D were not square-free, $D = f^2 d_K$, then H would be the ring class field of conductor f over K.) The equivalence between isomorphism classes of elliptic curves over $\bar{\mathbf{Q}}$ with endomorphism ring equal to R 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 such an endomorphism ring. Since our goal is to find such an elliptic curve modulo the prime p, it suffices to find a root j of $H_D(X)$ modulo p. Then the required elliptic curve is recovered as the curve with Weierstrass equation (assume $p \neq 2, 3$)

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

where

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

3 Our algorithm and its complexity

Before describing our algorithm, we introduce some notation. Assume that D is a fundamental discriminant and let $d = |D|$. Let $\mathcal{O}$ denote the ring of integers in $K = \mathbf{Q}(\sqrt{D})$ and let h denote the class number of K. Then asymptotically, h can be approximated by $\sqrt{d}$ (see [Lang1]). Denote the Hilbert class field of K by H. Let B be an upper bound on the size of the coefficients of $H_D(X)$.

In our complexity analysis, we assume that if a and b are two integers, then their addition takes time $O(\log a + \log b)$, their multiplication takes time $O(\log a \log b)$, and the division of the greater by the smaller takes time $O(\log a \log b)$. 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.

3.1 Overview of the algorithm

Let n be a positive integer. 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: According to [Cohen, §5.4], this can be done in time $O(d^{1/4})$, or in time $O(d^{1/5})$ assuming the generalized Riemann Hypothesis.

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

Version A:

(a) Generate a collection of distinct primes $\{p_i\}$, each satisfying $4p_i = t_i^2 + d$, for some integer t_i. Generate enough primes p_i so that the product of all the primes p_i exceeds the bound B (or maybe $2B$, see the remark after Example 7.1). Call the resulting set of primes S.
(b) For each p_i in S, count the number of points on a representative of each $\overline{\mathbf{F}}_p$ isomorphism class of elliptic curves. 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 j-invariants over F_p. We then form the set S_i consisting of all the j-invariants such that the corresponding curve has $p_i + 1 + t_i$ or $p_i + 1 - t_i$ points. There are exactly h such j values, by Prop. 4.1 and Prop. 4.2 below (or by [Cox, p. 319]).

The best implementations of elliptic curve point-counting algorithms currently run in time $O((\log p_i)^5)$. This step is repeated p_i times, so this step will take time $O(p_i(\log p_i)^5)$.
(c) For each prime p_i in S, we form the polynomial $H_D(X) \bmod p_i$ by multiplying together the factors $(X - j)$, where j is in the set S_i. This is justified by Prop. 4.1 and Prop. 4.2. The number of terms in the product is h and each coefficient is between zero and p_i, so this can be done in time $O(h^2(\log p_i)^2)$.

Version B:

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 $\{p_i\}$ of the form $4p_i = t_i^2 + u_i^2 d$, for some integers t_i and u_i. We again generate enough primes p_i so that their product exceeds the bound B; call the resulting set of primes T. Then for each p_i in T, we compute the endomorphism ring of each $\overline{\mathbf{F}}_p$ isomorphism class of elliptic curves using the algorithm in [Kohel]. We use the same representatives for the isomorphism classes as in Version A, Step (b) above, and we let j run through all possible j-invariants over F_p. We then form the set T_i consisting of all the j-invariants such that the corresponding curve has endomorhism ring isomorphic to, $\mathcal{O}$, the ring of integers of $\mathbf{Q}(\sqrt{D})$. The class number of $\mathcal{O}$ is h, so there are exactly h such j values.

If we assume the generalized Riemann hypothesis for K (p. 52 of [Kohel]), then Kohel's algorithm runs in the time $\mathrm{O}(p_i^{1/3+\epsilon})$ (Theorem 1 of [Kohel]) We have to repeat this step for each j-invariant, so this will take time $\mathrm{O}(p_i^{4/3})$ [or without GRH: $\mathrm{O}(p_i^2)$].

Remark 3.1. Note that when allowing the more general primes of the form $4p_i = t_i^2 + u_i^2 d$, where $u_i > 1$, we cannot just use point-counting to find the collection of elliptic curves that we want. If we did, then we would find by point-counting all the elliptic curves with endomorphism ring equal to any order in $\mathcal{O}$ containing the order of index u_i. In this paper, we have assumed that d is square-free from the beginning, but to generalize our algorithm to non-square-free d, 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 7.2). The main advantage to Version A is that it is easy to implement because there are many point-counting packages available. We used a version available on the web by Mike Scott for our examples. The main advantage of Version B is that it will generalize to work for all d.

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

Use the modified chinese remainder algorithm of Section 5 to compute each coefficient of $H_D(X) \bmod n$ using the values of the coefficients of $H_D(X) \bmod p_i$ computed in Step (1) for all the p_i's. The running time of this step is analyzed in Section 5. This step can be parallelized.

3.2 Complexity of the Algorithm

We would like to have an estimate for the size of B, an upper bound for the size of the coefficients of the class polynomial. As is explained in [AtMor, p. 18], we may take

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

where the sum in the above expression 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. This bound is the product of the roots times the largest binomial coefficient. To estimate 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 the number of divisors of a. This observation leads to the estimate

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

which is quoted in Crandall and Pommerance. Since the middle binomial coefficient is clearly less than 2^h, which is the sum of all of the binomial coefficients, we see that

$$B \approx 2^h e^{c(\log d)^2 \sqrt{d}},$$

for some constant c. So throughout the paper, we use the estimate

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

An important consideration for accurately assessing the running time of our algorithm is the relative size of the small primes found in Step (1). However, estimates for the size and number of the primes used in Step (1) are conjectural. A special case of the Bateman-Horn conjecture would imply that the size of the largest prime (of the special form $4p_i = t^2 + d$) required so that the product excedes B is roughly $(\log B)^2$, and there is a corresponding estimate for the number of primes required. For the purpose of comparing the two versions of the Chinese Remainder algorithm, we will use the model of the distribution of all primes, so that the largest prime can be expected to be roughly $\log B$ and the number of primes is estimated to be roughly $\log B/\log\log B$. Neither one of these frameworks fits our actual data from small examples very well (see Examples 7.1 and 7.2 of the Appendix).

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 p, as proposed in [CNST, §4], then the complexity of this procedure would be $\mathrm{O}(h(\log B)^2(\log\log B))$, which is not as good as our method in Step (2), which takes time $\mathrm{O}(h(\log p)(\log B)(\log\log B))$ when the discriminant is large (at least $(\log p)^2$), since $\log B = \mathrm{O}(d^{1/2}(\log d)^2)$). On the other hand, for primality proving as in [AtMor], one wants a small discriminant; in fact, then $d = \mathrm{O}((\log p)^2)$ (see [LL, §5.10]). 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 $\log p$, i.e., if the coefficients of $H_D(X)$ are large compared to p (which is quite likely).

4 Computing $H_D(X) \bmod p$ for small primes p

In this section, we prove that Step 1 of our algorithm is a valid way to compute $H_D(X) \bmod p$. 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 Hilbert class polynomial. Recall that H denotes the Hilbert class field of $\mathbf{Q}(\sqrt{D})$ and $\mathcal{O}$ is the ring of integers of $\mathbf{Q}(\sqrt{D})$. Let p be a rational prime that splits completely in H, i.e., splits into principal ideals in $\mathbf{Q}(\sqrt{D})$, i.e., $4p = t^2 - Du^2$ for some integers u and t. Splitting in H ensures that the reduction modulo a prime over p gives an elliptic curve over $\mathbf{F}_p$, as opposed to an extension of $\mathbf{F}_p$.

Let $\mathrm{Ell}(D)$ denote the set of isomorphism classes of elliptic curves over $\mathbf{C}$ with complex multiplication by $\mathcal{O}$ (i.e., whose ring of endomorphisms over $\mathbf{C}$ is isomorphic to $\mathcal{O}$). Then an equivalent way of defining the Hilbert class polynomial is as follows (see Section 2):

$$H_D(X) = \prod_{[E]\in \mathrm{Ell}(D)} (X - j(E)), \tag{2}$$

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

Let β be a prime ideal of the ring of integers of H lying over p. If E is an elliptic curve over $\mathbf{C}$ with complex multiplication by $\mathcal{O}$, then E is defined over H (in fact, $j(E)$ generates H over K, see [Silv2]). Suppose all such curves E have good reduction modulo β. This will happen for all but finitely many primes since there are finitely many elliptic curves as above, and each has finitely many primes of bad reduction. So without loss of generality we make this assumption. We denote the reduction of E at β by $\widetilde{E}$ (which is defined over $\mathbf{F}_p$). Then, by [Lang, Chap. 13,Thm. 12], $\widetilde{E}$ is ordinary. From the defintion of $j(E)$ in terms of the coefficients of the Weierstrass equation of E, it is easy to see that

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

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

Proposition 4.1. *With notation and assumptions as above,*

$$H_D(X) \bmod p = \prod_{[E']\in \mathrm{Ell}'(D)} (X - j(E')). \tag{3}$$

Proof. Let S denote the set of elliptic curves over $\mathbf{C}$ with complex multiplication by $\mathcal{O}$, and let S' denote the set of elliptic curves over $\mathbf{F}_p$ with complex multiplication by $\mathcal{O}$. First, by [Lang, Chap 13, Thm. 12(ii)] (or [Cox, Thm. 14.16]), reduction modulo β gives a map from S to S'. 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., we have an induced injective map from $\mathrm{Ell}(D)$ to $\mathrm{Ell}'(D)$. By the Deuring lifting theorem [Lang, Chap. 13, Thm. 14] (or [Cox, Thm. 14.16]) this map is also a surjection.

□

Proposition 4.2. *Suppose p is a prime and $x \le 2\sqrt{p}$ is an integer such that $4p = x^2 - D$ and D is square-free. 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 the trace of the Frobenius endomorphism of E'. Then $t = x$ or $t = -x$. In either case, the discriminant of the characteristic polynomial of the Frobenius endomorphism is $t^2 - 4p = x^2 - 4p = D$. Let $\mathrm{End}(E')$ denote the endomorphism ring of E'. Since D is square-free, the subring R of $\mathrm{End}(E')$ generated by the Frobenius endomorphism is $\mathcal{O}$, and at the same time $\mathrm{End}(E')$ is contained in the ring of integers of the quotient field of R. Hence $\mathrm{End}(E') = \mathcal{O}$, i.e., $[E'] \in \mathrm{Ell}'(D)$.

Conversely, suppose $[E'] \in \mathrm{Ell}'(D)$, and let t denote the the trace of the Frobenius endomorphism of E'. Suppose the Frobenius endomorphism generates a subring of index u in $\mathrm{End}(E')$, the endomorphism ring of E'. Then the characteristic polynomial of the Frobenius endomorphism has discriminant u^2D, hence $4p = t^2 - u^2D$. 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 $p+1+x$. $\square$

5 A modification of the Chinese remainder theorem

This section follows closely [Couv, §2.1], 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

$$\begin{aligned} M &= \prod_i m_i && (4)\\ M_i &= \prod_{j\neq i} m_j = M/m_i && (5)\\ a_i &= 1/M_i \bmod m_i, \qquad 0 \le a_i < m_i. && (6)\end{aligned}$$

Then the number $z = \sum_i a_i M_i x_i$ is congruent to x modulo M. Hence, if we round z/M to an integer r, then we have $x = z - rM$. 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. We have $r = \left[\frac{z}{M} + \frac{1}{2}\right]$. 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

$$\frac{z}{M} = \sum_i \frac{a_i x_i}{m_i}, \qquad (7)$$

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 $\gcd(a, b)$.

Modified Chinese Remainder Algorithm

From the discussion above, we obtain the following algorithm:

(i) Compute a_i's, for each i, using (6): this takes time $\mathrm{O}(\sum_i(\sum_j(\log m_j \log m_i) +$

$\ell(\log m_i)^2 + (\log m_i)^2)) = \mathrm{O}((\log M)^2 + \ell \sum (\log m_i)^2)$. [Reduce each m_j modulo m_i, take product and invert mod m_i, for each i.]
(ii) Compute $\mathrm{rem}(M, n)$ using (4): 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 (7), every term in the sum has to be calculated to precision ϵ/ℓ, hence the calculation of each term takes time $\mathrm{O}((\log(\ell/\epsilon))^2)$; we will assume that ϵ is picked to be an arbitrarily small number, say 0.01, thus making the overall calculation probabilistic (we can do several checks to see if r is wrong: e.g., the constant term of $H_D(X)$ is a cube, etc.). Then the calculation of all the terms in (7) will take total time $\mathrm{O}(\ell(\log \ell)^2)$ and the addition in (7) of ℓ numbers with precision ϵ/ℓ will take time $\mathrm{O}(\ell \log \ell)$.
(v) Output

$$\mathrm{rem}(x, n) = \mathrm{rem}\left(\left(\mathrm{rem}\left(\sum_i (\mathrm{rem}(a_i \cdot x_i, n) \cdot \mathrm{rem}(M_i, n)), n\right) - \mathrm{rem}(r, n) \cdot \mathrm{rem}(M, n)\right), n\right). \quad (8)$$

Complexity of the Modified Chinese Remainder Algorithm

The time taken for the various substeps in this step is as follows:
(a) Calculation of $\mathrm{rem}(a_i \cdot x_i, n)$ 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 (8) 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 $\mathrm{O}((\log n) \log(\sum m_i) + (\log n)^2)$ [reduce r mod n and then multiply by rem(M,n)].
(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 to lift $H_D(X) \bmod p_i$ for many p_i's to $H_D(X) \bmod p$ one coefficient at a time. Note that steps (i), (ii), and (iii) above are common to the lifting of every coefficient, and only step (iv) and (v) have to be repeated for each coefficient.

In the notation of Section 3, $m_i = p_i$. Using this, we see that the the time taken (asymptotically) for calculations in steps (v)(a), (v)(b) and (v)(c), performed h times, (once for each coefficient of $H_D(X)$) will dominate all other steps. We first calculate the overall complexity of these 3 steps using the estimates $\ell \approx \log B$ and $p_i \approx \log B$. This is true for the set of primes with no conditions of the type imposed in Step (1)(a) of our algorithm. In that case the complexity would be:

$$\mathrm{O}((\log p \log B \log \log B + (\log p + \log B)(\log p + \log \log B))),$$

and this needs to be repeated h times. This expression can be simplified using the fact that $\log \log B < \log p$, yielding

$$\mathrm{O}(\log p(\log B)(\log \log B) + (\log p)^2),$$

which should be performed h times. Compare this to the complexity for the usual Chinese remainder algorithm given below. Also note that, in addition, most of the steps in the algorithm above can be parallelized and even the lifting of the coefficients can be parallelized.

Comparison with usual Chinese Remainder Algorithm

If we are to use the naive Chinese remainder theorem for the problem stated at the beginning of this section, then we calculate

$$z = \mathrm{rem}\Big(\Big(\sum_i a_i \cdot x_i \cdot M_i\Big), M\Big), \tag{9}$$

and then reduce z modulo n.

The steps involved as are follows:
(i) Compute a_i's, for each i, using (6): 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 (9): 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 (9): takes time $\mathrm{O}((\log M)\log(\ell m_i^2 M))$.
(vi) Reducing z modulo n: takes time $\mathrm{O}((\log M)(\log n))$.

In the context of lifting $H_D(X) \bmod p_i$ to $H_D(X)$ and then reducing $H_D(X)$ modulo n, only steps (ii) – (vi) have to be repeated for each coefficient. Again in the notation of Section 3, $m_i = p_i$. For the sake of comparison, again assume that ℓ and p_i are $\mathrm{O}(\log M)$, and $M = B$. Thus the time taken for the calculation of step (ii) above, performed h times (once for each coefficient of $H_D(X)$), will dominate to give a complexity of $\mathrm{O}((\log B)^2(\log \log B))$ to be performed h times.

From this analysis, it is clear that our modified Chinese Remainder algorithm will be more efficient than the usual one whenever $\log p < \log B$, which will certainly be the case in our context whenever $d \geq (\log p)^2$.

6 Extensions

1) Instead of using just small primes, we could use powers of very small primes, where point-counting algorithms have recently been optimized.

2) Instead of counting points separately for each j-invariant using versions of Schoof's algorithm for each prime p_i, we could count points on all curves simultaneously using naive methods, making tables, and using match and sort algorithms. This would have the advantage of making the algorithm more elementary.

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.

[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.

[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.

[Tate] Tate, J., *Endomorphisms of abelian varieties over finite fields*, Invent. Math. 2 1966 134–144.

7 Appendix: Examples

In this section we present several examples to illustrate our algorithm.

7.1 $d = -59$

7.1.1 Atkin-Morain algorithm

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:

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

Here 28 decimal digits of precision are required using the package pari (19 digits of precision are not enough). Note: compare this to the value given in previous papers on this subject [BSS] [Cohen] [AtMor] [CNST], which estimates the amount of precision required to be about 63 (note that e^{63} translates to about 28 digits).

7.1.2 Modified Chinese Remainder algorithm

To implement our algorithm for this example, we set the bound B equal to e^{41} to be bigger than the largest coefficient of $h(x)$. 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 split completely in the Hilbert class field of $\mathbf{Q}(\sqrt{-59})$:

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

whose product excedes B. For each prime in the list p_i, we loop through the $p_i - 1$ possible j-values. For each possible j-value, we count the number of points on a curve over $\mathbf{F}_{p_i}$ with that j-value using a version of Schoof's algorithm. If the curve has either $p_i + 1 + t_i$ or $p_i + 1 - t_i$ points, with $t_i^2 = 4p_i - 59$, then we keep that j-value in a list S. At the end of the loop, we will have h j-values in the list S, where h is the degree of $h(x)$. Then the polynomial $h(x) \mod p_i$ is formed as the product over $j \in S$ of $(x - j)$.

Here is a table summarizing the results for this example:

Table 1:

p_i	t_i	$j \in S$	$h(x) \mod p_i$
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 $h(x)$ with integer coefficients using the usual Chinese Remainder Theorem. It takes as input the coefficients of $h(x)$ modulo the small primes p_i.

```
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 $h(x) \mod n$. Here is a short routine in Pari to compute the polynomial $h(x)$ 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. Suppose $n = 141767$ and we want a curve over the field of n elements with $142521 = n + 1 + 753$ points. The output is:

$$[Mod(31177, 141767), Mod(73152, 141767), Mod(48400, 141767)],$$

which corresponds to the class polynomial 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 7.1. 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. In general the best way to handle the problem is to require the product of the small primes to slightly exceed $2B$ (by an amount depending on your choice of epsilon).

7.2 $d = -832603$

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 another example with class number equal to 96. It is a larger example where the integer n has size approximately 28 bits, but where it is still more efficient to find the curve by counting points on all curves over $\mathbf{F}_n$ until one with the right number of points is found. Let

$$n = 100959557,$$

and suppose we want a curve over the field of n elements with $N = 100979633 = n + 1 + 20075$ points. In this case, $t = -20075$, so

$$d = -832603.$$

In this case, 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}.

7.2.1 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
$5474956970365974518712304022211825509601280000x^{95}$
...
$+$ 7322299210991241448561057156469098826736791832449631827456388132960112450371 93
29135064953296517214548131689852913646208744965072895238904299575578214929402556
90269474650890676517843149848915317444229033517941029021437170526121340304823144
94529405836883805689462371596200150318098817534799324978162644245655959842749947
17858432607385461461503717141914671818268157763154369005958127984642576929015204
26437854421276163389137659795914219505688231282732907305722902771611725883861541
48937250401909262758202693759197954966153758767383082942870504208104169346156761
37370140972768025144585055433107301880817078326024764664156908637120651225964726
46651744831906486468261619247046701837670500579714961113437036970935277571503153
22567537950401373183111375556463416615517590048068381995902769488747899512826899
14217610909882739179049340568049228656914511431462880918313022279161258449285248
67800549391010129865271636233396829450528957342703682099709542914419338084785528
15515541443389645541073956603508062130075042008584656621391660596796840702715468
37654086103968642152649693295050636361965872199680392706578180233023981945146803
36513644389585529149421547948017686720979681079328724693033766087188871609346000
09280626830932358147474239868774316904983332234331112135651989494795095128025242
84004442908640064547280302180916705994924904462408598493685783049510051195414203
74193714952172202764122207412399697680743188853363771174846147611536227962842763
30179212535126718469400321621422016180649215673515485858876699496218198280821008
67509106738774000289760625373465473138686010425839954970747220569129395699294677
20310519020112138986891840602925915129828885984248296412731268804138821857550046
31740535407261170371548562217188525436170617994265480900210744289591077201557084
43351878207996949375975214520812353864477617014941897442336805806919833270404356
72233745871756761220231883111379747450484770428379432492432112981852372370825200
73979346938165522772190403709801883425487153495041939758974438852349256849332582
42949120000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000x
$+$ 7774398957345170975803732914364873254397019842859308520183177973468286635919 50
63955335627487216215839225017162508128812823531148421047722257294972477891080734
80364913521716399773556190157658862923693124279517474658853246967319757166687213
19753733564349879126074104002426788016280120377306489828880425499653567931580120
74967899363537089145796582223376311022575164017505985077492038886558517805909831
01718325102837209142776392612949091540248366671439592152960504598152789962436827
68242622951943380283171051391738616321869756168188060284578707127840580209669688
20597900094300008626184027605852867394654243059909523444290639760800566332691660
73314080081952700898836046592306338322033669182801761007974687313404494147890694

```
72343180417359545258951738172149281447638743252884990224419538097825517357308859
65471506678389365003848838210004765548106160824576593239796003207769293084320623
61386480245925977557790037074423948680178235256276757154285930896131992560868715
57931457047920289686916138119438096974293017027075388754338662929943483912469915
85270293948158523396968975086805301827116473140098288830677454730497385971618505
71768357101193011042981967902715486884007865558967843641950597269806738937887969
06352088783830779545031722801487799435083344711059495588344883437966270131619905
14396274312932483473713897538532145621034257346800111972017915137738724875841542
15109682611174063164260663324610747358041227407041282400314489274231186441705126
28220393840767595121389551853420782935971325113194976518690123276819536081684512
58752839636530767215859943351643566275807889144879440625152095370714120641101342
85393396261926311021564254986125351824131379779331093359741475021197559687803375
56751490484820896604829816969815957321744265867298496962232629258793904307692458
00296808100840441856319640636442232282490965084808442230294032943893944311879931
53554600313110411121746414111770328198356802926696158261543756816692085932174387
55675418611108673895539866297504421991231914035953717719144705629257031124779008
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
```

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.

7.2.2 Modified Chinese Remainder Algorithm

To obtain the class polynomial using our method, we first make a large enough list of primes so that we can use the Chinese Remainder Theorem.

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 excedes the bound B as desired.

To illustrate the algorithm, we find the class polynomial for the largest prime on the list $p = 1434707$. 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 $p+1\pm\alpha$ points over $\mathbf{F}_p$, where $\alpha = 2215$ (note that $\alpha^2-d = 4p$):

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]

So the class polynomial for $p = 1434707$ is

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

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

The class polynomials modulo the rest of the 409 primes are not included here. Note that for our algorithm to be of interest, n should be much bigger.

Remark 7.2. In this example, we find that allowing primes of the more general form $4p_i = u_i^2 + v_i^2 d$ would not lead to any improvement of the algorithm. The size of v_i is constrained by the desire to keep the primes small; here, v_i must satisfy $v_i \leq 2$ to avoid getting larger primes. Allowing $v_i = 2$, 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 it does not look like Version B of this algorithm is necessary in practice.