Factoring Class Polynomials over the Genus Field

Marcel Martin
m.martin@ellipsa.eu

March 7, 2010

Abstract

Primality proving... Cryptography... As soon as we want to build an elliptic curve with a known order over a $\mathbb{Z}/p$ field using the so-called *complex multiplication*, we have to find a root of a class polynomial. Depending on the degree of this polynomial (and on the size of the prime p), this operation might be very lengthy. More concretely, suppose we have to find a root of $H_{-12932920}(x)$ (the degree of this polynomial is 832). Suppose now we can compute a factor of degree 13 more quickly than we can compute the whole polynomial $H_{-D}(x)$ itself. Of course, it would make the task easier...

Keywords: complex multiplication, genus field, class polynomial.

Notations

$\mathbb{Z}$ The rational integers.

$\mathbb{Q}$ The rational numbers.

$\mathbb{R}$ The real numbers.

$\mathbb{C}$ The complex numbers.

K The imaginary quadratic field $\mathbb{Q}(\sqrt{-D})$ where $-D$ is a negative fundamental discriminant.

$\mathcal{O}_K$ The maximal order of K, $\mathcal{O}_K = \begin{cases} \mathbb{Z}[\sqrt{-D}/2], & \text{if } D \equiv 0 \pmod 4 \\ \mathbb{Z}[(1+\sqrt{-D})/2], & \text{otherwise} \end{cases}$.

$\mathbb{Z}[x]$ The polynomials with coefficients in $\mathbb{Z}$

$\mathbb{Q}[x]$ The polynomials with coefficients in $\mathbb{Q}$

$\mathcal{O}_K[x]$ The polynomials with coefficients in $\mathcal{O}_K$

$\#(F)$ The number of elements in the set F (or the size of the vector F).

(a_i) The tuple $(a_0, \ldots, a_i, \ldots)$

$(a_i)_N$ The tuple $(a_0, \ldots, a_{N-1})$.

$\times$ The dot product operator.
For instance, $(a_i)_N \times (b_i)_N = (a_i * b_i)_N$.

$\oplus$ The **xor** (bitwise exclusive **or**) operator.
For instance, $5 \oplus 9 = 12$, i.e., $\overline{0101} \oplus \overline{1001} = \overline{1100}$ with a binary representation.

Introduction

Let $h(-D)$ and $g(-D)$, denoted h and g in the sequel, be the class number and the genus number associated with a negative fundamental discriminant $-D$. The class number h is the number of primitive reduced forms (a, b, c) of discriminant $-D = b^2 - 4ac$. The genus number g is the number of genera associated with $-D$ (see, for instance, [2, pp. 221–230] or [3, pp. 53–63]).

Our goal is to build the factors of a class polynomial $H_{-D}(x)$ [1] (when, of course, these factors exist, i.e., when $g > 1$ [2]) over a compositum of quadratic fields called the *genus field*. The genus field, denoted G_K in the sequel, is a field extension of $K = \mathbb{Q}(\sqrt{-D})$ [3]. More precisely, we want to obtain

$$H_{-D}(x) = \prod_{i=0}^{g-1} Q_i(x)$$

$$\text{with} \quad Q_i(x) = \frac{1}{g} \sum_{j=0}^{\frac{h}{g}-1} \left(\sum_{k=0}^{g-1} S_{i,k}\, B_k\, M_{k,j} \right) x^j + x^{\frac{h}{g}} \tag{1}$$

where S is a sign matrix (its coefficients are ± 1 [4]), B is a basis and M is a coefficient matrix (each of its column vectors consists of the coefficients of an integer of G_K multiplied by g). Note that the matrix S and the basis B could be merged into a single matrix equal to $S * \text{Diag}(B)$. They are not mainly for computational convenience: not only does the representation used minimize the memory needed to store the values, but B is not the same over $\mathbb{Z}/p$ as it is over $\mathbb{R}$ whereas S is the same for both cases.

Let H_K be the splitting field of $H_{-D}(x)$ [5]. $H_{-D}(x) \in \mathbb{Z}[x]$ being monic and irreducible, G_K being a subfield of H_K such that $[H_K : G_K] = h/g$, not only the factorization (1) exists when $g > 1$ but the $Q_i(x)$'s are conjugate polynomials over G_K, i.e., the g coefficients of degree j of the $Q_i(x)$'s are the g conjugates of an integer of G_K.

Before going further, let us see the purpose of the sign matrix S with a small (and artificial) example. Let $L = \mathbb{Q}(\sqrt{2}, \sqrt{3})$. L is a field extension of $\mathbb{Q}$ containing all the numbers of the form $u = a + b\sqrt{2} + c\sqrt{3} + d\sqrt{6}$ with $a, b, c, d \in \mathbb{Q}$. By definition, the Galois group $\text{Gal}(L/\mathbb{Q})$ consists of all the field automorphisms $\sigma : L \to L$ that fix $\mathbb{Q}$, i.e., such that $\sigma(q) = q$ for any $q \in \mathbb{Q}$. L being a Galois extension (because L is the splitting field of the separable polynomial $(x^2 - 2)(x^2 - 3) \in \mathbb{Q}[x]$), there are exactly $[L : \mathbb{Q}] = 4$ such automorphisms:

- σ_0, the identity map on L,
- σ_1, that takes $\sqrt{2}$ to $-\sqrt{2}$ and that fixes $\sqrt{3}$,
- σ_2, that takes $\sqrt{3}$ to $-\sqrt{3}$ and that fixes $\sqrt{2}$,
- σ_3, equal to $\sigma_1 \circ \sigma_2$.

[1] In the sequel, $H_{-D}(x)$ is used to denote any class polynomial, i.e., not the Hilbert ones only.

[2] If $g = 1$, there is a single factor, $H_{-D}(x)$ itself.

[3] The field G_K is the maximal unramified extension of K which is an Abelian extension of $\mathbb{Q}$.

[4] As we will see at Step 5, S is a Hadamard matrix of order g.

[5] The field H_K, called the *Hilbert class field* of K, is the maximal unramified Abelian extension of K.

With these σ_k's, we can compute the conjugates of u. We get [6]

$$\begin{aligned}\sigma_0(u) &= a + b\sqrt{2} + c\sqrt{3} + d\sqrt{6},\\ \sigma_1(u) &= a - b\sqrt{2} + c\sqrt{3} - d\sqrt{6},\\ \sigma_2(u) &= a + b\sqrt{2} - c\sqrt{3} - d\sqrt{6},\\ \sigma_3(u) &= a - b\sqrt{2} - c\sqrt{3} + d\sqrt{6}.\end{aligned}$$

In a matrix form, with $u = (a, b, c, d)$, i.e., with u expressed with respect to the basis $B' = (1, \sqrt{2}, \sqrt{3}, \sqrt{6})$, the previous equalities can be written

$$\begin{pmatrix}\sigma_0(u)\\ \sigma_1(u)\\ \sigma_2(u)\\ \sigma_3(u)\end{pmatrix} = \begin{pmatrix}+ & + & + & +\\ + & - & + & -\\ + & + & - & -\\ + & - & - & +\end{pmatrix} * \begin{pmatrix}1 & 0 & 0 & 0\\ 0 & \sqrt{2} & 0 & 0\\ 0 & 0 & \sqrt{3} & 0\\ 0 & 0 & 0 & \sqrt{6}\end{pmatrix} * \begin{pmatrix}a\\ b\\ c\\ d\end{pmatrix}.$$

Clearly, the line vectors of the sign matrix, let us call it S', consist of all the conjugates of the number $(1, 1, 1, 1)$ expressed with respect to the basis B'. Though the representation depends on the basis used, these lines represent the field automorphisms of $L/\mathbb{Q}$ [7] and they allow to compute any of the conjugates of any number $(a, b, c, d) \in L$, expressed with respect to B', simply by doing dot products. For instance, with $v = (1, -2, 3, -4)$,

$$\sigma_2(v) = (+, +, -, -) \times (1, -2, 3, -4) = (1, -2, -3, 4).$$

In the equation (1), the matrix S represents the field automorphisms of G_K/K [8] exactly like S' represents the ones of $L/\mathbb{Q}$. Being given $w = (a_0, \ldots, a_{g-1})$, a number of G_K expressed with respect to the basis B, the matrix S allows to get all the conjugates of w.

In the following sections, we will see how, for a given discriminant $-D$, one can compute the basis B and the matrix S ; then, how, by using them as well as floating point approximations of the $Q_i(x)$'s, one can get the matrix M, and, finally, how, with B, S and M, one can express the $Q_i(x)$'s over G_K or over $\mathbb{Z}/p$ fields. As numeric examples, all along the eight steps, we will use the data obtained with $-D = -2184$ for which we have $h = 24$ and $g = 8$.

[6] Instead of $\sigma(x)$, mathematicians often make use of σx or even of x^σ.

[7] In fact, the line vectors of S', with the dot product operation, are a group isomorphic to Gal$(L/\mathbb{Q})$ (we obviously have $S'_{\sigma_i} \times S'_{\sigma_j} = S'_{\sigma_i \circ \sigma_j}$ for any $\sigma_i, \sigma_j \in$ Gal$(L/\mathbb{Q})$).

[8] It should be noted that Gal(G_K/K) is not a subset (let alone a subgroup) of Gal(H_K/K). As a matter of fact, Gal(H_K/G_K) being a normal subgroup of Gal(H_K/K), Gal(G_K/K) is isomorphic to the quotient group Gal(H_K/K)/Gal(H_K/G_K) or, equivalently, to Gal$_K(H_{-D}(x))$/Gal$_{G_K}(Q_i(x))$.

Step 1: Factoring the discriminant

A negative discriminant $-D$ is *fundamental* if D is a positive integer not divisible by the square of an odd prime and if $D \equiv 3 \pmod 4$ or $D \equiv 4, 8 \pmod{16}$.

First of all, we have to compute a small table F containing all the prime factors, possibly signed, of $-D$ (F may also contain -1). Though this is not the way it is implemented, the factorization is simple: for all p's that are odd prime factors of D, we put $p^* = (-1)^{(p-1)/2}\, p$ in F, then we divide $-D$ by the product of all the p^*'s and the remaining even factor, if any, is divided by 4 and added to F.

For computational convenience, the first part of F, denoted F^- in the sequel, contains the negative factors in decreasing order (this part is never empty); the second part contains the positive factors in increasing order.

Algorithm 1.1 *(Factoring the discriminant)*

input

 D, absolute value of a fundamental discriminant (small integer)

outputs

 F, array of factors (small integers)

 g, genus number associated with $-D$ (small integer)

begin

 $i \leftarrow 0$

 $j \leftarrow 0$

 if $(D \text{ mod } 16) = 4$ ***then***

  $F_0 \leftarrow -1$

  $i \leftarrow 1$

  $D \leftarrow D/4$

 elseif $(D \text{ mod } 32) = 8$ ***then***

  $F_0 \leftarrow -2$

  $i \leftarrow 1$

  $D \leftarrow D/8$

 elseif $(D \text{ mod } 32) = 24$ ***then***

  $T_0 \leftarrow 2$ *// T is a temporary table*

  $j \leftarrow 1$

  $D \leftarrow D/8$

 endif

 while $D > 1$ ***do*** *// here D is a square-free product of odd primes*

  $p \leftarrow$ *smallest prime factor of* D

  $D \leftarrow D/p$

  if $(p \text{ mod } 4) = 3$ ***then***

   $F_i \leftarrow -p$

   $i \leftarrow i + 1$

  else

   $T_j \leftarrow p$

$j \leftarrow j + 1$
endif
endwhile
for k ***from*** 0 ***upto*** $j - 1$ ***do*** $F_{i+k} \leftarrow T_k$ *// append T to F*
$g \leftarrow 2^{i+j-1}$
end

With $-D = -2184$, we have $-D = -8 * -3 * -7 * 13$, so we get $F = (-2, -3, -7, 13)$ and $g = 8$.

Using F, we can describe the genus field of $K = \mathbb{Q}(\sqrt{-2184}) = \mathbb{Q}(\sqrt{-546})$ as an extension of $\mathbb{Q}$: $G_K = \mathbb{Q}(\sqrt{-2}, \sqrt{-3}, \sqrt{-7}, \sqrt{13})$, as well as an extension of K: $G_K = K(\sqrt{-3}, \sqrt{-7}, \sqrt{13})$. In the latter case, there are several possible descriptions. In the sequel, we implicitly make use of $G_K = K(\sqrt{6}, \sqrt{14}, \sqrt{13})$. The real extensions allow to build a basis B, for G_K/K, that has no imaginary parts (this is, of course, better from a computational point of view).

Step 2: The basis

In order to express the integers of G_K as tuples of coefficients, we need a basis. For our purposes, this basis consists of the square roots of the g possible positive products $\prod_{i=0}^{\#(F)-1} F_i^{e_i}$ where the e_i exponents are in $\{0, 1\}$.

The algorithm 2.1 is the fastest one I found (there are only $g - \#(F) + \left\lfloor 2^{\#(F^-)-2} \right\rfloor$ multiplications) to compute these products while producing the whole sequence so that (3) holds.

Algorithm 2.1 *(Computing the basis)*
inputs
    F, *factors of* $-D$ *(array of small integers)*
    g, *genus number of* $-D$ *(small integer)*
output
    A, *dot square of the basis (array[g] of small integers)*
begin
    $A_0 \leftarrow 1$
    $k \leftarrow 1$
    if *Size*(F^-) > 1 ***then***
        for i ***from*** 1 ***upto*** *Size*(F^-) $- 1$ ***do***
            $A_k \leftarrow F_i$
            for j ***from*** 1 ***upto*** $k - 1$ ***do*** $A_{j+k} \leftarrow A_j * A_k$ ***endfor***
            $k \leftarrow k * 2$
        endfor
        for j ***from*** 1 ***upto*** $k - 1$ ***do***
            if $A_j < 0$ ***then*** $A_j \leftarrow A_j * F_0$ ***endif***
        endfor
    endif
    for i ***from*** *Size*(F^-) ***upto*** *Size*(F) $- 1$ ***do***
        $A_k \leftarrow F_i$
        for j ***from*** 1 ***upto*** $k - 1$ ***do*** $A_{j+k} \leftarrow A_j * A_k$ ***endfor***
        $k \leftarrow k * 2$
    endfor
end

With $F = (-2, -3, -7, 13)$, we get $A = (1, 6, 14, 21, 13, 78, 182, 273)$ and the basis is $B = (1, \sqrt{6}, \sqrt{14}, \sqrt{21}, \sqrt{13}, \sqrt{78}, \sqrt{182}, \sqrt{273})$.

With the composition law $\odot$ defined as

$$A_i \odot A_j = \frac{A_i * A_j}{\gcd(A_i, A_j)^2}\,, \tag{2}$$

the set of the A_k values is a group isomorphic to $(\mathbb{Z}/2\mathbb{Z})^t$ with $t = \log_2(g)$, i.e., isomorphic to $(\{0, 1, ..., g-1\}, \oplus)$, and, by construction, A is ordered such that

$$A_i \odot A_j = A_{i \oplus j}\,. \tag{3}$$

Since $A_i * A_j = (A_i \odot A_j) * \gcd(A_i, A_j)^2$ and since $D \equiv 0 \pmod{A_k}$, the previous equality implies

$$\text{Jacobi}(A_i * A_j, n) = \text{Jacobi}(A_{i \oplus j}, n) \tag{4}$$

for any n such that $n > 1$ and $\gcd(n, 2D) = 1$.

Note that we could also build A using the field description $G_K = K(\sqrt{6}, \sqrt{14}, \sqrt{13})$ as a seed. First, we would set $A_1 = 6$, $A_2 = 14$ and $A_4 = 13$ (i.e., we would set all the A_{2^i}'s), then we would use (3) in order to compute the other A_k's [9]. From a computational point of view, the algorithm 2.1 is more efficient, especially when working over $\mathbb{Z}/p$ fields as we will have to do at Step 8 (we will have no modular inverses to compute), but both methods lead to the same results.

[9] This is about the way D. Chatelain defines the sequence (A_i), see [1, §7.3.1].

Step 3: List of primitive reduced forms

A binary quadratic form is a polynomial $ax^2 + bxy + cy^2 \in \mathbb{Z}[x, y]$ denoted (a, b, c) for short. Its discriminant is $-D = b^2 - 4ac$.
A form is *primitive* if $\gcd(a, b, c) = 1$. When its discriminant is negative, a form is *positive definite* if $a > 0$ and $c > 0$ and it is *reduced* if $|b| \leq a \leq c$ and if $b \geq 0$ whenever $a = c$ or $a = |b|$. The set of the h positive definite, primitive and reduced binary quadratic forms of negative discriminant $-D$, with a law called *composition of forms* (see [4, § 5.6.3]) and denoted $\circ$, is a commutative group called the *class group* and denoted $\mathcal{C}(-D)$ in the sequel. The *principal form* is the identity element of $\mathcal{C}(-D)$. A form is *ambiguous* if it is its own inverse. The ambiguous forms are of the types $(a, 0, c)$, (a, a, c) or (a, b, a).

The algorithm 3.1 fills up a list L with $(h + g)/2$ positive definite, primitive and reduced forms (a, b, c) of discriminant $-D$. There are g forms of L that are ambiguous. Since we only store a form (a, b, c) and not its inverse $(a, -b, c)$, the remaining $(h - g)/2$ forms are half of all the non-ambiguous forms.

Algorithm 3.1 *(Generating the forms)*

input
  D, *absolute value of a fundamental discriminant (small integer)*
outputs
  g, *genus number (small integer)*
  h, *class number (small integer)*
  L, *list of primitive reduced forms* (a, b, c) *(small integers)*
    $b^2 - 4ac = -D$ *for each* L_i
begin
  $bmax \leftarrow \left\lfloor \sqrt{D/3} \right\rfloor$
  $b \leftarrow D \bmod 2$
  $i \leftarrow 0$
  if $b = 0$ ***then***
    $a \leftarrow 1$
    $s \leftarrow 1$
    $t \leftarrow D/4$
    repeat
      if $(t \bmod a) = 0$ ***then***
        $L_i \leftarrow (a, 0, t/a)$
        $i \leftarrow i + 1$
      endif
      $s \leftarrow s + a$
      $a \leftarrow a + 1$
      $s \leftarrow s + a$ // $s = a^2$
    until $s > t$
    $b \leftarrow 2$
  endif

```
        g ← i
        while b ≤ bmax do
            a ← b
            s ← a * a
            t ← (s + D) / 4
            repeat
                if (t mod a) = 0 then
                    c ← t/a
                    L_i ← (a, b, c)
                    i ← i + 1
                    if (a = b) or (a = c) then g ← g + 1 endif
                endif
                s ← s + a
                a ← a + 1
                s ← s + a  // s = a^2
            until s > t
            b ← b + 2
        endwhile
        h ← i + i − g
    end
```

With $-D = -2184$, we obtain $h = 24$, $g = 8$, and the list of the table 1.

	(a, b, c)
L_0	$(1, 0, 546)$
L_1	$(2, 0, 273)$
L_2	$(3, 0, 182)$
L_3	$(6, 0, 91)$
L_4	$(7, 0, 78)$
L_5	$(13, 0, 42)$
L_6	$(14, 0, 39)$
L_7	$(21, 0, 26)$
L_8	$(5, 4, 110)$
L_9	$(10, 4, 55)$
L_{10}	$(11, 4, 50)$
L_{11}	$(22, 4, 25)$
L_{12}	$(15, 6, 37)$
L_{13}	$(17, 14, 35)$
L_{14}	$(19, 18, 33)$
L_{15}	$(23, 22, 29)$

Table 1: List of forms

In our example, the ambiguous forms are all located at the beginning of the list L. This is not always the case with other discriminants. The produced lists being sorted on b, ambiguous forms of the kind $a = b$ or $a = c$, if any, might be located anywhere.

Remark. To generate the primitive reduced forms one can also make use of the algorithm proposed in [7, §A.13.2] but note that, instead of fundamental discriminants $-D$, they make use of "reduced" discriminants $-d$ that are equal to either $-D$ or $-D/4$. The forms their algorithm produces are not always the same than the ones produced by the algorithm 3.1. With P1363 forms, the discriminant $-d$ is equal to $b^2 - ac$, not to $b^2 - 4ac$.
It is easy to get one value knowing the other one:

- ***if*** $(d \bmod 4) = 3$ ***then*** $D \leftarrow d$ ***else*** $D \leftarrow 4d$ ***endif***
- ***if*** $(D \bmod 4) = 0$ ***then*** $d \leftarrow D/4$ ***else*** $d \leftarrow D$ ***endif***

Step 4: Weighting the genera

An integer n is *represented* by a form (a, b, c) if $ax^2 + bxy + cy^2 = n$ for some integers (x, y). A *genus* is a set of forms. The *principal genus*, denoted $\mathcal{G}_0(-D)$ (or simply $\mathcal{G}_0$), is a subgroup of the class group $\mathcal{C}(-D)$ constituted of all the "squares" $(a, b, c) \circ (a, b, c)$. The other genera are cosets of $\mathcal{G}_0$.
G_K/K being an unramified Abelian extension, the *Artin symbol* $(\frac{G_K/K}{\mathfrak{n}})$, where $\mathfrak{n}$ is a fractional ideal of $\mathcal{O}_K$ prime to $2D$, is an element of $\mathrm{Gal}(G_K/K)$. Denoted φ in the sequel, this symbol uniquely identifies the genus of a form and it can be represented with a tuple of signs (see [9, §4.2.2]) describing its action on the elements of G_K.

For each form (a, b, c) of the list L, we compute an integer n such that $n > 1$ [10], $\gcd(n, 2D) = 1$ and n is represented by the form. With non-ambiguous forms, we compute n only for (a, b, c) since n is also represented by $(a, -b, c)$ [11]. Then we compute $\varphi_{(a,b,c)} \simeq (\mathrm{Jacobi}(F_i, n))_{\#(F)}$ and we give it a weight w such that $0 \leq w < g$.

Algorithm 4.1 *(Weighting the genera)*
inputs
    F, *factors of* $-D$ *(array of small integers)*
    g, *genus number of* $-D$ *(small integer)*
    h, *class number of* $-D$ *(small integer)*
    L, *list of* $(h+g)/2$ *primitive reduced forms (3-tuples of small integers)*
output
    W, *weights (array[*$(h+g)/2$*] of small integers)*
begin
    for i ***from*** 0 ***upto*** $(h+g)\ /\ 2 - 1$ ***do***
        $n \leftarrow$ *integer such that* $n > 1$, $\gcd(n, 2D) = 1$ *and* n *represented by* L_i
        $W_i \leftarrow 0$
        for j ***from*** *Size*$(F) - 1$ ***downto*** *Size*(F^-) ***do***
            $W_i \leftarrow W_i * 2$
            if *Jacobi*$(F_j, n) < 0$ ***then*** $W_i \leftarrow W_i + 1$ ***endif***
        endfor
        if *Size*$(F^-) > 1$ ***then***
            u $\leftarrow$ *Jacobi*(F_0, n)
            for j ***from*** *Size*$(F^-) - 1$ ***downto*** 1 ***do***
                $W_i \leftarrow W_i * 2$
                if *Jacobi*$(F_j, n) \neq u$ ***then*** $W_i \leftarrow W_i + 1$ ***endif***
            endfor
        endif
    endfor
end

[10] With an "extended" Jacobi symbol, i.e., such that $Jacobi(any, 1) = 1$, we could allow $n = 1$.
[11] If n is represented by (a, b, c) with (x, y), it is represented by $(a, -b, c)$ with $(x, -y)$ or $(-x, y)$.

In order to obtain the φ tuples, by using the *Kronecker symbol* (see [4, p. 212]) instead of the Jacobi symbol in the algorithm 4.1, we could avoid to compute the integers n represented by the forms since, for any i such that $0 \leq i < \#(F)$,

$$\varphi_{(a,b,c),\,F_i} = \begin{cases} \text{Kronecker}(F_i, a), & \text{if} \begin{cases} ((F_i = -1) \ \text{ and } \ (a \text{ is odd})) \\ \text{or} \\ (a \not\equiv 0 \pmod{F_i}) \end{cases} \\ \text{Kronecker}(F_i, c), & \text{if} \begin{cases} ((F_i = -1) \ \text{ and } \ (c \text{ is odd})) \\ \text{or} \\ (c \not\equiv 0 \pmod{F_i}) \end{cases} \end{cases}.$$

The form (a, b, c) being primitive, if $F_i = -1$, a and c cannot be both even, and, if $F_i \neq -1$, a and c cannot be both divisible by F_i.

	(a,b,c)	(x,y)	n	J(-2,n)	J(-3,n)	J(-7,n)	J(13,n)	weight
L_0	$(1,0,546)$	$(1,1)$	547	+	+	+	+	0
L_1	$(2,0,273)$	$(1,1)$	275	+	−	+	−	5
L_2	$(3,0,182)$	$(1,1)$	185	+	−	−	+	3
L_3	$(6,0,91)$	$(1,1)$	97	+	+	−	−	6
L_4	$(7,0,78)$	$(1,1)$	85	−	+	+	−	7
L_5	$(13,0,42)$	$(1,1)$	55	−	+	−	+	1
L_6	$(14,0,39)$	$(1,1)$	53	−	−	+	+	2
L_7	$(21,0,26)$	$(1,1)$	47	−	−	−	−	4
L_8	$(5,4,110)$	$(1,0)$	5	−	−	−	−	4
L_9	$(10,4,55)$	$(0,1)$	55	−	+	−	+	1
L_{10}	$(11,4,50)$	$(1,0)$	11	+	−	+	−	5
L_{11}	$(22,4,25)$	$(0,1)$	25	+	+	+	+	0
L_{12}	$(15,6,37)$	$(0,1)$	37	−	+	+	−	7
L_{13}	$(17,14,35)$	$(1,0)$	17	+	−	−	+	3
L_{14}	$(19,18,33)$	$(1,0)$	19	+	+	−	−	6
L_{15}	$(23,22,29)$	$(1,0)$	23	−	−	+	+	2

Table 2: List of weighted forms

With $F = (-2, -3, -7, 13)$, the algorithm 4.1 produces the values reported in the table 2. In this table, we see that each genus (the forms having the same weight w are in the same genus) contains exactly one ambiguous form. This is not always the case with other discriminants. In fact, there is one ambiguous form in each genus if and only if h/g is odd (see [11, p. 44]).

The table 3 shows how the genera are weighted. In the table 2, the first Jacobi symbol column, which is always associated with F_0, is combined (dot product) with all other columns associated with negative F_i's. If, for some discriminant, only F_0 is negative, then the first column is simply ignored. Doing so, the J$(*, n)$'s of the table 3 header are always $(\text{Jacobi}(A_{2^i}, n))$ with the A_{2^i}'s computed at Step 2. Then we replace the value x of each cell by $(1 - x)/2$ and we get the binary expressions of the weights. More concisely, we define the weight w of each genus as being

$$w = \bigoplus_{i=0}^{log_2(g)-1} \left(\frac{1 - \text{Jacobi}(A_{2^i}, n)}{2} * 2^i \right). \tag{5}$$

J(6,n)	J(14,n)	J(13,n)	bit 0	bit 1	bit 2	weight
+	+	+	0	0	0	0
−	+	−	1	0	1	5
−	−	+	1	1	0	3
+	−	−	0	1	1	6
−	−	−	1	1	1	7
−	+	+	1	0	0	1
+	−	+	0	1	0	2
+	+	−	0	0	1	4

Table 3: Weights

It is clear that we can simplify the algorithm 4.1 by computing directly the Jacobi symbol values with the A_{2^i}'s rather than with the F_i's. Note that, in that case, we can no more make use of the Kronecker symbol as explained above (even with a primitive form (a, b, c), $(\gcd(A_{2^i}, a) \neq 1)$ and $(\gcd(A_{2^i}, c) \neq 1)$ may simultaneously occur).

Algorithm 4.2 *(Weighting the genera)*

inputs

A, "squared" basis (array[g] of small integers)

g, genus number of $-D$ (small integer)

h, class number of $-D$ (small integer)

L, list of $(h + g)/2$ primitive reduced forms (3-tuples of small integers)

output

W, weights (array[$(h + g)/2$] of small integers)

begin

for i **from** 0 **upto** $(h + g) / 2 - 1$ **do**

$n \leftarrow$ *integer such that* $n > 1$, $\gcd(n, 2D) = 1$ *and* n *represented by* L_i

$j \leftarrow g$

$W_i \leftarrow 0$

while $j > 1$ **do**

$j \leftarrow j/2$

$W_i \leftarrow W_i * 2 + 1 - \text{Jacobi}(A_j, n)$

endwhile

$W_i \leftarrow W_i/2$

endfor

end

Let $\mathcal{G}_w$ be the genus of weight w and let N_w be any n such that $\gcd(n, 2D) = 1$, $n > 1$ and n is represented by a form of $\mathcal{G}_w$. For instance, using the table 2, N_3 could be equal to 185 or to 17 (as a matter of fact, N_w represents an equivalence class containing an infinity of values).

Using N_w, let us rewrite (5) as

$$w = \bigoplus_{i=0}^{log_2(g)-1} \left(\frac{1 - \text{Jacobi}(A_{2^i}, N_w)}{2} * 2^i \right). \tag{6}$$

We get

$$\text{Jacobi}(A_{2^i}, N_{2^j}) = \begin{cases} -1, & \text{if } i = j \\ +1, & \text{if } i \neq j \end{cases}, \tag{7}$$

and

$$\text{Jacobi}(A_i, N_j) = \text{Jacobi}(A_j, N_i). \tag{8}$$

(6) ⇒ (7)
It is sufficient to replace w by 2^j in (6). All the terms of the right hand side should be equal to 0 except the one for which $i = j$.

(6) ⇒ (8)
Let $\mathcal{B}_{i,k}$ be equal to $(1 - \text{Jacobi}(A_{2^k}, N_i))/2$ (so, $\mathcal{B}_{i,k} \in \{0, 1\}$) and let us rewrite (6) as

$$i = \bigoplus_{k=0}^{log_2(g)-1} \left(2^k * \mathcal{B}_{i,k}\right).$$

Since $A_0 = 1$, we obviously have $A_{2^k * \alpha} = A_{2^k}^{\alpha}$ whenever $\alpha \in \{0, 1\}$. Using this, as well as the previous equality and the equality (4), we get

$$\begin{aligned} \text{Jacobi}(A_i, N_j) &= \text{Jacobi}\left(A_{\bigoplus_{k=0}^{log_2(g)-1}(2^k * \mathcal{B}_{i,k})}, N_j\right) \\ &= \text{Jacobi}\left(\Pi_{k=0}^{log_2(g)-1} A_{2^k * \mathcal{B}_{i,k}}, N_j\right) \\ &= \prod_{k=0}^{log_2(g)-1} \text{Jacobi}\left(A_{2^k * \mathcal{B}_{i,k}}, N_j\right) \\ &= \prod_{k=0}^{log_2(g)-1} \text{Jacobi}\left(A_{2^k}^{\mathcal{B}_{i,k}}, N_j\right) \\ &= \prod_{k=0}^{log_2(g)-1} \text{Jacobi}\left(A_{2^k}, N_j\right)^{\mathcal{B}_{i,k}} \\ &= \prod_{k=0}^{log_2(g)-1} (1 - 2\mathcal{B}_{j,k})^{\mathcal{B}_{i,k}} \end{aligned}$$

and, since $(1 - 2\alpha)^{\beta} = (1 - 2\beta)^{\alpha}$ always holds with $(\alpha, \beta) \in \{0, 1\}^2$, the result follows.

Remark. In the sequel, rather than using the notation $\varphi_{(a,b,c)}$, we will use φ_w where w is the weight associated with the genus of the form (a, b, c).

Step 5: The sign matrix

Let $u = (1, 1, \ldots, 1)$ be an integer of G_K expressed with respect to the basis B. As said in the Introduction, the line vectors of the matrix S are equal to the conjugates of u, i.e., they are equal to $\{\varphi_w(u)\}_{0 \leq w < g}$ (recall that φ_w is an element of $\mathrm{Gal}(G_K/K)$).

φ_w being the map that sends $\sqrt{F_i}$ to $\mathrm{Jacobi}(F_i, N_w) * \sqrt{F_i}$ for all $i \in 0..\#(F) - 1$, we know its action on any $\sqrt{A_k}$. It comes

$$\varphi_w(u) = (\mathrm{Jacobi}(A_0, N_w),\ \mathrm{Jacobi}(A_1, N_w),\ \ldots,\ \mathrm{Jacobi}(A_{g-1}, N_w)).$$

So, building S is straightforward. We set the table 4 using our example $-D = -2184$. The most left column of the table contains the weights w. The four next columns reproduce the φ_w tuples obtained at Step 4. Except for $A_0 = 1$, the sign matrix (the $8 * 8$ matrix on the right of the table) was obtained by means of dot products. For instance, the column $A_5 = 78 = -2 * -3 * 13$ is simply the product of the columns -2, -3 and 13.

w	-2	-3	-7	13	1	6	14	21	13	78	182	273
0	+	+	+	+	+	+	+	+	+	+	+	+
1	−	+	−	+	+	−	+	−	+	−	+	−
2	−	−	+	+	+	+	−	−	+	+	−	−
3	+	−	−	+	+	−	−	+	+	−	−	+
4	−	−	−	−	+	+	+	+	−	−	−	−
5	+	−	+	−	+	−	+	−	−	+	−	+
6	+	+	−	−	+	+	−	−	−	−	+	+
7	−	+	+	−	+	−	−	+	−	+	+	−

Table 4: Sign matrix

The sign matrix has an important property: $g * S^{-1} = {}^tS$ (the transpose of S).
1. Since the lines of S represent the conjugates $\varphi_w(u)$ of $u = (1, 1, \ldots, 1)$ in G_K, their sum is equal to $\mathrm{Trace}(u)_{G_K/K} = (g, 0, \ldots, 0)$.
2. $\mathrm{Jacobi}(A_i * A_j, N_k) = \mathrm{Jacobi}(A_{i \oplus j}, N_k)$ (see Step 2).
So, if $c_{i,j}$ is a cell of ${}^tS * S$, we have

$$\begin{aligned} c_{i,j} &= \sum_{k=0}^{g-1} {}^tS_{i,k} * S_{k,j} = \sum_{k=0}^{g-1} S_{k,i} * S_{k,j} = \sum_{k=0}^{g-1} \mathrm{Jacobi}(A_i, N_k) * \mathrm{Jacobi}(A_j, N_k) \\ &= \sum_{k=0}^{g-1} \mathrm{Jacobi}(A_i * A_j, N_k) = \sum_{k=0}^{g-1} \mathrm{Jacobi}(A_{i \oplus j}, N_k) = \begin{cases} g, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases}. \end{aligned}$$

Thus ${}^tS * S = g * I_g$.

The previous equality shows that the sign matrix we get is particularly nice, it is a Hadamard matrix of order g (as a matter of fact, with our orderings on the weights and on the A_k's, it is even a Sylvester-Hadamard matrix since $S_0 = \varphi_0(u) = u = (1, 1, ..., 1)$ and, due to (8), $(S_w = \varphi_w(u)) \Rightarrow (S \text{ is symmetrical})$).

There are two interesting consequences:
1) We have no matrix to invert (the inverse we need at Step 7 is given for free).
2) Instead of building S using the φ tuples as explained above, we can take advantage of the recursive structure of Sylvester-Hadamard matrices.

So, assuming the table A is built as explained at Step 2 and assuming the genera are weighted as indicated at Step 4, for any discriminant $-D$, the sign matrix can be built with

$$S^{(1)} = (+)$$

and, for $t > 0$,

$$S^{(2^t)} = \begin{pmatrix} S^{(2^{t-1})} & S^{(2^{t-1})} \\ S^{(2^{t-1})} & -S^{(2^{t-1})} \end{pmatrix}.$$

Since we are interested in $S^{(8)}$ (for $-D = -2184$), here it is

$$S^{(8)} = \begin{pmatrix} S^{(4)} & S^{(4)} \\ S^{(4)} & -S^{(4)} \end{pmatrix}$$

$$= \begin{pmatrix} S^{(2)} & S^{(2)} & S^{(2)} & S^{(2)} \\ S^{(2)} & -S^{(2)} & S^{(2)} & -S^{(2)} \\ S^{(2)} & S^{(2)} & -S^{(2)} & -S^{(2)} \\ S^{(2)} & -S^{(2)} & -S^{(2)} & S^{(2)} \end{pmatrix}$$

$$= \begin{pmatrix} S^{(1)} & S^{(1)} & S^{(1)} & S^{(1)} & S^{(1)} & S^{(1)} & S^{(1)} & S^{(1)} \\ S^{(1)} & -S^{(1)} & S^{(1)} & -S^{(1)} & S^{(1)} & -S^{(1)} & S^{(1)} & -S^{(1)} \\ S^{(1)} & S^{(1)} & -S^{(1)} & -S^{(1)} & S^{(1)} & S^{(1)} & -S^{(1)} & -S^{(1)} \\ S^{(1)} & -S^{(1)} & -S^{(1)} & S^{(1)} & S^{(1)} & -S^{(1)} & -S^{(1)} & S^{(1)} \\ S^{(1)} & S^{(1)} & S^{(1)} & S^{(1)} & -S^{(1)} & -S^{(1)} & -S^{(1)} & -S^{(1)} \\ S^{(1)} & -S^{(1)} & S^{(1)} & -S^{(1)} & -S^{(1)} & S^{(1)} & -S^{(1)} & S^{(1)} \\ S^{(1)} & S^{(1)} & -S^{(1)} & -S^{(1)} & -S^{(1)} & -S^{(1)} & S^{(1)} & S^{(1)} \\ S^{(1)} & -S^{(1)} & -S^{(1)} & S^{(1)} & -S^{(1)} & S^{(1)} & S^{(1)} & -S^{(1)} \end{pmatrix}$$

$$= \begin{pmatrix} + & + & + & + & + & + & + & + \\ + & - & + & - & + & - & + & - \\ + & + & - & - & + & + & - & - \\ + & - & - & + & + & - & - & + \\ + & + & + & + & - & - & - & - \\ + & - & + & - & - & + & - & + \\ + & + & - & - & - & - & + & + \\ + & - & - & + & - & + & + & - \end{pmatrix}$$

Of course, the obtained matrix is equal to the matrix of the table 4.

Algorithm 5.1 *(Computing the sign matrix)*

input

  g, *genus number (small integer)*

output

  S, *sign matrix (array[g,g] of small integers (± 1))*

begin

  $S_{0,0} \leftarrow 1$

  $k \leftarrow 1$

  while $k < g$ ***do***

    for i ***from*** 0 ***upto*** $k-1$ ***do***

      for j ***from*** 0 ***upto*** $k-1$ ***do***

        $S_{i+k,\,j} \leftarrow S_{i,\,j}$

        $S_{i,\,j+k} \leftarrow S_{i,\,j}$

        $S_{i+k,\,j+k} \leftarrow -S_{i,\,j}$

      endfor

    endfor

    $k \leftarrow k * 2$

  endwhile

end

Note that $S_{i,\,j} = (-1)^{W(i,j)}$ where $W(i,j)$ is the Hamming weight of $(i \wedge j)$ [[12]]. It is not difficult to show it since, with $0 \leq i < 2^k$ and $0 \leq j < 2^k$, we always have

- $W(i+2^k, j) = W(i, j+2^k) = W(i,j)$,
- $W(i+2^k, j+2^k) = W(i,j)+1$.

[12] The symbol $\wedge$ indicates the bitwise **and** operator.

Step 6: Floating point approximations of the $Q_i(x)$'s

In order to compute a class polynomial, the easiest method consists in computing floating point approximations of its roots and to use them to build the polynomial. Though it is impossible to know in advance the exact precision required, there are rules to overestimate it. For the class invariants described in [7, §A.13.3] (these invariants are based on the Weber functions $\mathfrak{f}$, $\mathfrak{f}_1$ and $\mathfrak{f}_2$), one can use the rules proposed in [8, §4].

L_i	(a,b,c)	Associated root
L_0	$(1, 0, 546)$	$+33012526.575181343491\ldots$
...		
L_{11}	$(22, \pm 4, 25)$	$+0.07856366481761975165 4\ldots$ $\pm 0.53094269739398157020\ldots\ i$
...		

Table 5: Roots

We compute the roots associated with all the forms of the list L. Then, we build g polynomials $T_k(x)$, of degree h/g, by regrouping the roots according to the genus of their associated forms. These polynomials, that always have real coefficients [13], are stored in an array $T[0..g-1,\ 0..h/g-1]$. We don't store the leading coefficients of the polynomials since they are always equal to 1. In the array T, the line vector $T[k, \ldots]$ contains the coefficients of the polynomial built with the roots associated with the forms of $\mathcal{G}_k$. For instance, with $-D = -2184$, using the roots associated with the three forms of $\mathcal{G}_0$ (see Table 5), we get

$T_0(x) = T_{0,0} + T_{0,1}\ x + T_{0,2}\ x^2 + x^3$

where

$T_{0,0} = -9509997.67294691\ldots$
$T_{0,1} = +5187170.43334302\ldots$
$T_{0,2} = -33012526.7323087\ldots$

Note that the choice of $\mathcal{G}_i$ to get the roots of $T_i(x)$ is somewhat arbitrary. The use of the basis B and of the sign matrix S, as they are defined at Step 2 and Step 5, implies that we have to associate the sequences

$$(\mathcal{G}_i,\ \mathcal{G}_{i\oplus 1},\ \ldots,\ \mathcal{G}_{i\oplus(g-1)}) \mapsto (T_0(x),\ T_1(x),\ \ldots,\ T_{g-1}(x))\ [14]$$

for some $i \in 0..g-1$ but we can take any i. The g possible choices lead to g different permutations of the $T_k(x)$'s, i.e., to g different matrices M in the equation (1). For instance, if $\mathcal{G}_0 \mapsto T_0(x)$ leads to $M^{(0)}$ then $\mathcal{G}_k \mapsto T_0(x)$ (or, equivalently, $\mathcal{G}_0 \mapsto T_k(x)$) leads to $M^{(k)} = \mathrm{Diag}(S_k) * M^{(0)}$, where $\mathrm{Diag}(S_k)$ is the diagonal matrix built with the $(k+1)$-th line vector of S (this line represents the field automorphism φ_k).

[13] Assuming that, like here, we use a class invariant such that two conjugate roots are associated with two forms belonging to a same genus.

[14] This is a consequence of both Galois Theory and Class Field Theory.

Step 7: The coefficient matrix

When expressed with respect to a basis B as computed at Step 2, the real integers of G_K have coefficients [15] that are not in $\mathbb{Z}$ but in $(\frac{1}{g})\mathbb{Z}$, so we can write them as $\frac{1}{g}\sum_{i=0}^{g-1} a_i B_i$ where the a_i's are in $\mathbb{Z}$. It is the reason why there is a $\frac{1}{g}$ factor in the equation (1). The column vectors of M are coefficients of integers of G_K multiplied by g so that they are integers and not fractions.

Identifying the coefficients (except the leading ones) of the factors $Q_i(x)$ with the array T computed at Step 6, let us rewrite the equation (1) in a matrix form. It comes

$$\frac{1}{g} * S * \text{Diag}(B) * M = T$$

$$\frac{1}{g} * M = \text{Diag}(B)^{-1} * S^{-1} * T$$

$$M = \text{Diag}(B)^{-1} * {}^tS * T \quad (\text{using } g * S^{-1} = {}^tS)$$

and, since we are working with floating point approximations and not with exact values, we finally get

$$M_{i,j} = \text{Round}\left(\frac{1}{B_i}\sum_{k=0}^{g-1} S_{k,i}\, T_{k,j}\right)$$

Algorithm 7.1 *(Computing the matrix M)*
inputs
  C, *basis over* $\mathbb{R}$ *(inverted, i.e.,* $C_i = 1/\sqrt{A_i}$*) (array[g] of big reals)*
  g, *genus number (small integer)*
  h, *class number (small integer)*
  S, *sign matrix (array[g,g] of small integers (*± 1*))*
  T, *matrix computed at Step 6 (array[g,*h/g*] of big reals)*
output
  M, *coefficient matrix (array[g,*h/g*] of big integers)*
begin
  for i ***from*** 0 ***upto*** $g-1$ ***do***
    for j ***from*** 0 ***upto*** $h/g-1$ ***do***
      $x \leftarrow T_{0,j}$ *// we know that* $S_{0,i} = 1$
      for k ***from*** 1 ***upto*** $g-1$ ***do*** $x \leftarrow x + S_{k,i} * T_{k,j}$ ***endfor***
      $M_{i,j} \leftarrow$ *Round(*$x * C_i$*)*
    endfor
  endfor
end

Of course, all the operations with the floating point numbers x and C_i's should be done using the precision found at Step 6.

[15] The $Q_i(x)$ polynomials having real coefficients, we only need the real integers of G_K.

With our example, $-D = -2184$, we get a matrix M equal to

$$\begin{pmatrix} -9509688 & 5187192 & -33012360 \\ -3882456 & 2117808 & -13477368 \\ -2541664 & 1386432 & -8823008 \\ -2075184 & 1131936 & -7203888 \\ -2637584 & 1438560 & -9155984 \\ -1076832 & 587328 & -3737952 \\ -704952 & 384496 & -2447064 \\ -575568 & 313920 & -1998000 \end{pmatrix}$$

Here, even if we did not (implicitly) multiply the coefficients by g, they would have been integers since they are all divisible by $g = 8$ but this is not always the case with other discriminants.

At this point, we have all we need in order to express the $Q_i(x)$'s of (1). For instance

$$\begin{aligned} Q_4(x) &= \frac{1}{8}\sum_{j=0}^{2}\left(\sum_{k=0}^{7} S_{4,k}\, B_k\, M_{k,j}\right) x^j + x^3 \\ &= Q_{4,0} + Q_{4,1}\, x + Q_{4,2}\, x^2 + x^3 \end{aligned}$$

where

$$\begin{aligned} Q_{4,0} &= -\,1188711 - 485307\sqrt{6} - 317708\sqrt{14} - 259398\sqrt{21} \\ &\quad + 329698\sqrt{13} + 134604\sqrt{78} + 88119\sqrt{182} + 71946\sqrt{273} \\ Q_{4,1} &= +\,648399 + 264728\sqrt{6} + 173304\sqrt{14} + 141492\sqrt{21} \\ &\quad - 179820\sqrt{13} - 73416\sqrt{78} - 48062\sqrt{182} - 39240\sqrt{273} \\ Q_{4,2} &= -\,4126545 - 1684671\sqrt{6} - 1102876\sqrt{14} - 900486\sqrt{21} \\ &\quad + 1144498\sqrt{13} + 467244\sqrt{78} + 305883\sqrt{182} + 249750\sqrt{273} \end{aligned}$$

Remark. The $Q_i(x)$ polynomials being conjugate over G_K, to quickly get any coefficient $Q_{k,j}$ from $Q_{i,j}$ (expressed as tuples of coefficients with respect to the basis B), it is sufficient to make the dot products $Q_{i,j} \times S_i \times S_k$ where S_n is the (n+1)-th line vector of the matrix S. In fact, due to the ordering we are using since the beginning, we can also compute it with $Q_{k,j} = Q_{i,j} \times S_{i \oplus k}$. For instance,

$$\begin{aligned} Q_{3,0} &= Q_{4,0} \times S_7 \\ &= Q_{4,0} \times (+,-,-,+,-,+,+,-) \\ &= (-1188711, 485307, 317708, -259398, -329698, 134604, 88119, -71946) \end{aligned}$$

or, else,

$$\begin{aligned} Q_{3,0} = &-\,1188711 + 485307\sqrt{6} + 317708\sqrt{14} - 259398\sqrt{21} \\ &- 329698\sqrt{13} + 134604\sqrt{78} + 88119\sqrt{182} - 71946\sqrt{273}. \end{aligned}$$

Step 8: Working over $\mathbb{Z}/p$ fields

The factorization (1) is also valid over any $\mathbb{Z}/p$ field assuming p is a prime such that $\gcd(p, 2D) = 1$ and $4p = x^2 + Dy^2$ for some $(x, y) \in \mathbb{Z}^2$.
In the sequel, we will use the prime $p = \frac{1418446^2 + 2184 * 809283^2}{4} = 358099677116323$ to go on with our example $-D = -2184$.

Computing a basis over $\mathbb{Z}/p$ is not as straightforward as it is over $\mathbb{R}$. Due to the way we compute it, we have to count the number W_i of negative factors, except F_0, used for each B_i and to "negate" B_i when $W_i \equiv 2, 3 \bmod 4$.
Note that W_i is simply the Hamming weight of i.

Algorithm 8.1 *(Computing a basis over $\mathbb{Z}/p$)*
inputs
  p, odd prime such that $4p = x^2 + Dy^2$ for some $(x, y) \in \mathbb{Z}^2$ (big integer)
  R, square roots mod p of the F_i's (array[$\#(F)$] of big integers)
    R_0 is required only if $\#(F^-) > 1$
output
  B, basis over $\mathbb{Z}/p$ (array[g] of big integers)
begin
  $B_0 \leftarrow 1$
  $k \leftarrow 1$
  if *Size*$(F^-) > 1$ ***then***
    for i ***from*** 1 ***upto*** *Size*$(F^-) - 1$ ***do***
      $B_k \leftarrow R_i$
      $W_k \leftarrow 1$ *// W, local array[$1..2^{\#(F^-)-1} - 1$] of small integers*
      for j ***from*** 1 ***upto*** $k - 1$ ***do***
        $B_{j+k} \leftarrow (B_j * R_i)$ *mod* p
        $W_{j+k} \leftarrow W_j + 1$
      endfor
      $k \leftarrow k * 2$
    endfor
    for j ***from*** 1 ***upto*** $k - 1$ ***do***
      if *odd*(W_j) ***then*** $B_j \leftarrow (B_j * R_0)$ *mod* p ***endif*** *// because, here, $A_j \leftarrow A_j * F_0$*
      if $(W_j$ *mod* $4) \geq 2$ ***then*** $B_j \leftarrow p - B_j$ ***endif*** *// "negate" B_j*
    endfor
  endif
  for i ***from*** *Size*(F^-) ***upto*** *Size*$(F) - 1$ ***do***
    $B_k \leftarrow R_i$
    for j ***from*** 1 ***upto*** $k - 1$ ***do*** $B_{j+k} \leftarrow (B_j * R_i)$ *mod* p ***endfor***
    $k \leftarrow k * 2$
  endfor
end

With $-D = -2184$, $p = 358099677116323$ and the following list of roots

$R_0 = 1124906542557$
$R_1 = 193124827720271$
$R_2 = 82133832588560$
$R_3 = 345798145618681$

the algorithm 8.1 produces the basis

$B_0 = 1$
$B_1 = 138579447850272$
$B_2 = 195858486873162$
$B_3 = 206988590703680$
$B_4 = 345798145618681$
$B_5 = 203082986192536$
$B_6 = 316722244248718$
$B_7 = 289064347795142$

It should be noted that the obtained basis depends on the R_i's (for instance,using $p - R_0$ instead of R_0 would have produced the basis $(B_0, p - B_1, p - B_2, B_3, B_4, p - B_5, p - B_6, B_7)$).
Of course, whatever is the basis, among the possible ones, that will be used with the algorithm 8.3, we will get the same $Q_i(x)$'s. The only difference is that they will not be produced in the same order.

At Step 7, we implicitly multiplied the coefficients of the matrix M by g. We can now cancel this operation, done in order to get integers and not fractions, by dividing the coefficients of the basis by g modulo p. We divide the coefficients of the basis and not the ones of the matrix M simply because there is generally less work to do (the basis and the matrix contain respectively g and h coefficients and we always have $g \leq h$).

Algorithm 8.2 *(Dividing the basis by g modulo p)*

inputs
B, basis over $\mathbb{Z}/p$ (array[g] of big integers)
g, genus number of $-D$ (small integer)
p, odd prime (big integer)

output
B, basis over $\mathbb{Z}/p$ divided by g modulo p (array[g] of big integers)

begin
$k \leftarrow g$
while $k > 1$ ***do***
for i ***from*** 0 ***upto*** $g - 1$ ***do***
if odd(B_i) ***then*** $B_i \leftarrow B_i + p$ ***endif***
$B_i \leftarrow B_i/2$
endfor
$k \leftarrow k/2$
endwhile
end

With $-D = -2184$ and $p = 358099677116323$, we obtain
$B_0 = 223812298197702$
$B_1 = 17322430981284$
$B_2 = 114007230138226$
$B_3 = 25873573837960$
$B_4 = 267037066400037$
$B_5 = 25385373274067$
$B_6 = 308165038368332$
$B_7 = 304707801311635$

At last!

Algorithm 8.3 *(Computing one factor of $H_{-D}(x)$ over $\mathbb{Z}/p$)*
inputs
B, basis over $\mathbb{Z}/p$ divided by g modulo p (array of big integers)
g, genus number of $-D$ (small integer)
h, class number of $-D$ (small integer)
i, index of the wished factor (small integer in $0..g-1$)
M, coefficient matrix (array[g,h/g] of big integers)
p, odd prime such that $4p = x^2 + Dy^2$ for some $(x, y) \in \mathbb{Z}^2$ (big integer)
S, sign matrix (array[g,g] of small integers (± 1))
output
Q, polynomial of degree h/g, factor of $H_{-D}(x)$ over $\mathbb{Z}/p$
begin
$Q_{h/g} \leftarrow 1$
for j ***from*** 0 ***upto*** $h/g - 1$ ***do***
$Q_j \leftarrow B_0 * M_{0,j}$ *// we know that $S_{i,0} = 1$*
for k ***from*** 1 ***upto*** $g - 1$ ***do*** $Q_j \leftarrow Q_j + S_{i,k} * B_k * M_{k,j}$ ***endfor***
$Q_j \leftarrow Q_j$ *mod* p
endfor
end

With i running through $0..7$, the algorithm 8.3 produces the following polynomials modulo p

$Q_0(x) = x^3 + 349411664140631\,x^2 + 236118815942277\,x + 121688504601529$
$Q_1(x) = x^3 + 168320784679033\,x^2 + 99689071127264\,x + 274269421593516$
$Q_2(x) = x^3 + 280369563518210\,x^2 + 206498150577522\,x + 114371282890567$
$Q_3(x) = x^3 + 43259257063184\,x^2 + 213239562466426\,x + 19424900783462$
$Q_4(x) = x^3 + 142239847045079\,x^2 + 70353977608422\,x + 104800940627475$
$Q_5(x) = x^3 + 337091300899581\,x^2 + 128594135500790\,x + 251124699723139$
$Q_6(x) = x^3 + 221712819911823\,x^2 + 91086453164646\,x + 217088197522536$
$Q_7(x) = x^3 + 248093115311714\,x^2 + 28718870148814\,x + 329630751213380$

And each of these 8 polynomials is a factor, over $\mathbb{Z}/p$, of

$$\begin{aligned} H_{-2184}(x) = x^{24} &- 33012360\,x^{23} - 5499066444\,x^{22} - 38191097592\,x^{21} \\ &-860945475774\,x^{20} + 2860345968552\,x^{19} + 7390791596004\,x^{18} \\ &+18071068156632\,x^{17} + 49152082910703\,x^{16} + 73526500711728\,x^{15} \\ &+80616276081768\,x^{14} + 104922626382288\,x^{13} + 137712813694364\,x^{12} \\ &+104922626382288\,x^{11} + 80616276081768\,x^{10} + 73526500711728\,x^{9} \\ &+49152082910703\,x^{8} + 18071068156632\,x^{7} + 7390791596004\,x^{6} \\ &+2860345968552\,x^{5} - 860945475774\,x^{4} - 38191097592\,x^{3} \\ &-5499066444\,x^{2} - 33012360\,x + 1 \end{aligned}$$

When only one factor of $H_{-D}(x)$ is needed (for instance, in the context of an ECPP implementation), by merging the algorithms 7.1 and 8.3, we compute and make use of the coefficients of the matrix M without storing them. Note that, when $g > h/g$, it is better to divide by g not the basis but the coefficients (up to the degree $h/g - 1$) of the returned polynomial.

Algorithm 8.4 *(Computing a factor of $H_{-D}(x)$ over $\mathbb{Z}/p$ without matrix M)*

inputs

- B, *basis over $\mathbb{Z}/p$ divided by g modulo p (array of big integers)*
- C, *basis over $\mathbb{R}$ (inverted, i.e., $C_i = 1/\sqrt{A_i}$) (array of big reals)*
- g, *genus number of $-D$ (small integer)*
- h, *class number of $-D$ (small integer)*
- p, *odd prime such that $4p = x^2 + Dy^2$ for some $(x, y) \in \mathbb{Z}^2$ (big integer)*
- S, *sign matrix (array[g,g] of small integers (± 1))*
- T, *matrix computed at Step 6 (array[g,h/g] of big reals)*

output

- Q, *polynomial of degree h/g, factor of $H_{-D}(x)$ over $\mathbb{Z}/p$*

begin

    $Q_{h/g} \leftarrow 1$
    for j ***from*** 0 ***upto*** $h/g - 1$ ***do***
        $Q_j \leftarrow 0$
        for i ***from*** 0 ***upto*** $g - 1$ ***do***
            $x \leftarrow T_{0,j}$ *// we know that $S_{0,i} = 1$*
            for k ***from*** 1 ***upto*** $g - 1$ ***do*** $x \leftarrow x + S_{k,i} * T_{k,j}$ ***endfor***
            $Q_j \leftarrow Q_j + B_i *$ *Round*$(x * C_i)$
        endfor
        $Q_j \leftarrow Q_j$ *mod* p
    endfor
end

Benchmarks

Here are some timings obtained with $-D = -12932920$ ($h = 832, g = 64, h/g = 13$) using miscellaneous class invariants.

$Invariant$	P_1	T_1	P_2	T_2	T_1/T_2
$\mathfrak{f}^2/\sqrt{2}$	1824	2.06	768	0.31	6.65
$\mathfrak{w}_{13}^2$	5600	26.84	1344	1.35	19.88
$\mathfrak{w}_7^4$	9504	77.84	2528	4.29	18.14
$\mathfrak{w}_5^6$	12576	136.92	3520	8.38	16.34
γ_2	24384	398.06	5856	14.76	26.97
j	73120	4835.00	17504	170.81	28.31

Table 6: $-D = -12932920$

P_1: precision used to compute the polynomial $H_{-D}(x)$.
P_2: precision used to compute the matrix M.
T_1: time to build $H_{-D}(x)$.
T_2: time to build M (including the re-computation, done for the sake of comparison, of the list of roots with the smaller precision P_2 [[16]]).
The precisions are in bits, the times in seconds.

[16]Due to the difference between the required precisions, computing a factor over the genus field is almost always (much) faster than computing the whole class polynomial.

Conclusion

I have detailed a method that is not difficult to implement and that works well in practice [17]. This method can be seen as an extension, working when $h \geq g$ [18], of the method of D. Bernardi presented in [9, §6.2.3].

The only other method (that works when $h \geq g$) I am aware of is the one proposed by A. Atkin and F. Morain in [1, §7.3]. Having never implemented it, the only thing I can say is that it seems a little more complicated than the one I have presented. According to the authors, one has to solve a *generic system of order* g (this is a linear system).

Though I only considered polynomials that are elements of $\mathbb{Z}[x]$, it is not difficult to adapt the method so that it works when the polynomials are elements of $\mathcal{O}_K[x]$.

Acknowledgments

I am grateful to Michael Scott for his helpful comments on a preliminary version of this *how to*.

First publication (draft 0.1) February 20, 2006
Copyright © 2010, Marcel Martin

[17] My ECPP implementation, the software *Primo* [10], makes use of it since now nine years.

[18] The method of D. Bernardi only works when $h = g$, i.e., it only works with 56 discriminants (counting the ones such that $g > 1$) called *Euler numbers* or *idoneal numbers* (numeri idonei).

References

[1] A. O. L. ATKIN, F. MORAIN, **Elliptic curves and primality proving**, in *Math. Comp.* 61, 203, July 1993, pp. 29-68.
http://www.lix.polytechnique.fr/~morain/Articles/ecpp.ps.gz

[2] H. COHN, **Advanced Number Theory**, Dover Publications Inc., New York, 1980.

[3] D. COX, **Primes of the form $\mathbf{x^2 + ny^2}$**, John Wiley & Sons, Inc., 1989.

[4] R. CRANDALL, C. POMERANCE, **Prime numbers, a computational perspective**. 1st ed., Springer, 2001.

[7] **IEEE P1363 / D8 (Draft version 8)**. *Standard Specifications for Public Key Cryptography*. Annex A (informative). Number-Theoretic Background.
http://www.informatik.tu-darmstadt.de/TI/Veroeffentlichung/Artikel/Kryptographie/PublicKeyAllgemein/

[8] E. KONSTANTINOU, Y. STAMATIOU, C. ZAROLIAGIS, **On the Use of Weber Polynomials in Elliptic Curve Cryptography** in *Public Key Infrastructure: First European PKI Workshop, Research and Applications* - EuroPKI 2004, LNCS 3093, Springer, pp. 335–349, 2004.
http://students.ceid.upatras.gr/~konstane/papers/europki2004-weber-poly.pdf

[9] F. MORAIN, **Implementation of the Atkin-Goldwasser-Kilian Primality Testing Algorithm**, RR #911, INRIA, October 1988.
http://www.lix.polytechnique.fr/Labo/Francois.Morain/Articles/INRIA-RR911.ps.gz

[10] Primo - ECPP implementation.
http://www.ellipsa.eu/

[11] J. ROBERTSON, **Computing in Quadratic Orders**, 2006.
http://www.jpr2718.org/