On the Efficient Generation of Elliptic Curves over Prime Fields*

Elisavet Konstantinou[1,2], Yiannis C. Stamatiou[1,2], and Christos Zaroliagis[1,2]

[1] Computer Technology Institute, P.O. Box 1122, 26110 Patras, Greece
[2] Department of Computer Engineering and Informatics,
University of Patras, 26500 Patras, Greece
{konstane,stamatiu,zaro}@ceid.upatras.gr

Abstract. We present a variant of the complex multiplication method that generates elliptic curves of cryptographically strong order. Our variant is based on the computation of Weber polynomials that require significantly less time and space resources than their Hilbert counterparts. We investigate the time efficiency and precision requirements for generating off-line Weber polynomials and its comparison to another variant based on the off-line generation of Hilbert polynomials. We also investigate the efficiency of our variant when the computation of Weber polynomials should be made on-line due to limitations in resources (e.g., hardware devices of limited space). We present trade-offs that could be useful to potential implementors of elliptic curve cryptosystems on resource-limited hardware devices.

1 Introduction

Elliptic curve cryptography constitutes a fundamental and efficient technology for public key cryptosystems. One of the most important problems in elliptic curve cryptography is the generation of cryptographically secure elliptic curves over prime fields. One method to achieve this is by repeated applications of point counting [4]: select an elliptic curve (EC) at random, count its order (number of rational points on the curve), and check whether the order is *suitable*, that is, it satisfies certain conditions that guarantee cryptographic strength (i.e., resistance to known attacks). Unfortunately, this method can be extremely slow.

An alternative method which generates ECs of a suitable order is the *Complex Multiplication* (CM) method [1]. This method first determines a suitable order and then constructs an EC of that order. The input to the method is a prime p (representing the order of the prime field) from which the so-called *CM discriminant D* is computed. The EC is generated by constructing certain polynomials based on D and finding their roots.

* This work was partially supported by the IST Programme of EU under contracts no. IST-1999-14186 (ALCOM-FT) and no. IST-1999-12554 (ASPIS), and by the Human Potential Programme of EU under contract no. HPRN-CT-1999-00104 (AMORE).

There are two variations of Complex Multiplication, depending on whether Hilbert or Weber polynomials are used (see Section 4), that have two main differences: (i) Hilbert polynomials can have huge coefficients as the discriminant D increases, while for the same discriminant the Weber polynomials have much smaller coefficients and thus are easier to construct; (ii) the roots of the Hilbert polynomial construct directly the EC, while the roots of the Weber polynomial have to be transformed to the roots of its corresponding Hilbert polynomial to construct the EC.

When one considers hardware implementations of the CM method on embedded systems, one problem that immediately arises is that the Hilbert polynomials require high-precision floating point and complex arithmetic (i.e., large registers and floating point units) for their construction and storage. Thus, the Hilbert polynomials do not seem appropriate for hardware implementation that generates them on-line.

To alleviate this shortcoming of Hilbert polynomials, a variant of the CM method was recently proposed in [16] that turns out to be rather efficient. This variant takes as input the discriminant D and then computes the prime field's order p and the order m of the EC. The only condition for cryptographic strength posed on m is that it should be prime. Since Hilbert polynomials depend only on D, they can be precomputed off-line (for various values of D) and stored for subsequent use. Thus, if one wants to build an EC of a specific order (ensuring cryptographic strength) for a certain value of D, then one could simply index the stored Hilbert polynomial using D and, if succeeds in finding the desired curve's order, proceed with the next steps of the CM method.

Although the above variant tackles adequately the efficient construction of ECs, there may still be problems with storing and handling several Hilbert polynomials with huge coefficients on hardware devices (e.g., microcontroller chips) with limited resources. Since in such devices the size of floating-point units and the available memory for data and code are limited, it is desirable to keep their sizes as low as possible. It is perhaps because of this reason that (to the best of our knowledge) the vast majority of language tools developed for such hardware devices are based on ANSI C.

In this paper, we further investigate the space and time efficiency of the CM method by shifting our attention to the Weber polynomials. We present another variant of the CM method, similar to the one given in [16], that uses Weber polynomials. Our variant takes also as input the discriminant D, but selects the field's order p at random (or selects it from a set of prescribed primes) and subsequently computes the curve order m using a different method, requesting that m (is not necessarily prime but it) should satisfy the suitability conditions given in [4, Sec. V.7] for cryptographic strength. We have implemented our variant in ANSI C using the (ANSI C) library GNUMP [7]. Based on this implementation, we have conducted an experimental study over a large number of ECs investigating the precision requirements for the off-line generation of Weber polynomials in comparison with the generation of the corresponding Hilbert polynomials. We were also interested in investigating the efficiency of our variant that uses precomputed Weber polynomials in comparison to the variant in [16] that uses precomputed Hilbert polynomials, and the efficiency of our variant when constructing Weber polynomials on-line. The latter is of particular importance in space-limited hardware systems.

Our experiments showed that for a wide range of discriminants and polynomial degrees the construction of Weber polynomials requires significantly less time and precision than that required for the construction of the Hilbert polynomials. The experiments revealed a trade-off between the two CM variants depending on the values of D, the polynomial degree h, and the space availability of the hardware environment on which the CM method will be implemented. In particular, our experiments showed that, for several values of D and relatively small values of h, our CM implementation requires many fewer iterations in order to find a suitable curve order m and its time efficiency compares favorably with that reported in [16]. When both h and D are getting relatively large, however, our variant becomes less time efficient than the CM variant in [16] (mainly because of the different method for computing m and finding the roots of polynomials). Hence, if there is sufficient space availability for storing either type of precomputed polynomials, the CM variant in [16] seems beneficial for large values of D and h, while ours is better for smaller values of h. On the other hand, if there are space constraints, the storage of Hilbert polynomials for large values of D and h may not be possible. Our experiments showed that, for several values of D and relatively small values of h, the time of our CM implementation for generating an EC of a suitable order by computing *on-line* the Weber polynomials compares favorably with the time the CM variant in [16] takes to generate ECs of prime order using precomputed Hilbert polynomials. Since a small value of h does not necessarily imply a compromise in security, the on-line construction of Weber polynomials could be used in such cases as an alternative to the off-line construction of Hilbert polynomials. Even in the case where a larger value of h is required, it would be more space-efficient to precompute and store the Weber polynomials for the requested large values of h and compute on-line the Weber polynomials for the smaller values of h.

The rest of the paper is organized as follows. In Section 2, we state briefly some basic definitions and results from elliptic curve theory. In Section 3, we present the basic CM method and our variant, while in Section 4 we describe the construction of the Hilbert and the Weber polynomials, along with some examples aiming at the explanation of their computational requirements. In Section 5 we discuss some implementation related issues, while in Section 6 we discuss our experimental results. We conclude in Section 7.

2 Preliminaries of Elliptic Curve Theory

In this section we review some basic concepts regarding elliptic curves and their definition over finite fields. The interested reader may find additional information in e.g., [4,21]. We also assume familiarity with elementary number theory (see e.g., [5]).

An *elliptic curve* $E(F_p)$ over a finite field F_p, where $p > 3$ and prime, is the set of points $(x, y) \in F_p$ (represented by affine coordinates) which satisfy the equation

$$y^2 = x^3 + ax + b \tag{1}$$

and $a, b \in F_p$ are such that $4a^3 + 27b^2 \neq 0$. The set of solutions (x, y) of Eq. (1) together with a point $\mathcal{O}$, called the *point at infinity*, and a special addition operation define an Abelian group, called the *Elliptic Curve group.* The point $\mathcal{O}$ acts as the identity element (details on how the addition is defined can be found in e.g., [4,21]).

The *order* m of an elliptic curve is the number of the points in $E(F_p)$. The expression $t = p+1-m$ (which measures the difference between m and p) is called the *Frobenius trace* t. Hasse's theorem (see e.g., [4,21]) states that $|t| \leq 2\sqrt{p}$ which gives upper and lower bounds for m based on p:

$$p + 1 - 2\sqrt{p} \leq m \leq p + 1 + 2\sqrt{p}. \tag{2}$$

The *order of a point* P is the smallest positive integer n for which $nP = \mathcal{O}$. Application of Langrange's theorem (see e.g., [5]) on $E(F_p)$, gives that the order of a point $P \in E(F_p)$ always divides the order of the elliptic curve group, so $mP = \mathcal{O}$ for any point $P \in E(F_p)$, which in turn implies that the order of a point cannot exceed the order of the elliptic curve.

Two important quantities associated with $E(F_p)$ are the *curve discriminant* Δ and the j*-invariant*, defined by

$$\Delta = -16(4a^3 + 27b^2) \tag{3}$$

and

$$j = \frac{-1728(4a)^3}{\Delta} \tag{4}$$

Given a j-invariant $j_0 \in F_p$ ($j_0 \neq 0, 1728$), two elliptic curves can be easily constructed. The first EC is of the form defined by Eq. (1) and can be constructed by setting $a = 3k \bmod p$, $b = 2k \bmod p$, where $k = \frac{j_0}{1728 - j_0} \bmod p$. The second EC, called the *twist* of the first, is defined as

$$y^2 = x^3 + ac^2x + bc^3 \tag{5}$$

where c is any quadratic non-residue in F_p. If m_1 is the order of an EC and m_2 is the order of its twist, then $m_1 + m_2 = 2p + 2$, i.e., if one curve has order $p + 1 - t$, then its twist has order $p + 1 + t$, or vice versa [4, Lemma VIII.3].

The security of elliptic curve cryptosystems is based on the difficulty of solving the discrete logarithm problem (DLP) on the EC group. To ensure intractability of solving this problem by all known attacks, the group order m should obey the following conditions:

1. m must have a sufficiently large prime factor (greater than 2^{160}).

2. m must not be equal to p.
3. For all $1 \leq k \leq 20$, it should hold that $p^k \not\equiv 1 \pmod{m}$.

The first condition excludes the application of type of methods like the Pohlig-Hellman [14] one to solve DLP, the second condition excludes the application of the anomalous attack [15,20,22], while the third condition excludes the MOV attack [12]. If the order of an EC group satisfies the above conditions, we shall call it *suitable.*

3 The Complex Multiplication Method and Our Variant

The theory of complex multiplication (CM) of elliptic curves over the rationals can be used to generate elliptic curves of a suitable order m, resulting in the so-called *CM method*. The CM method computes j-invariants from which is then easy to construct the EC. The method is based on the following idea (for more details see [4,8]).

Hasse's theorem implies that $Z = 4p - (p+1-m)^2$ is positive. This in turn implies that there is a unique factorization $Z = Dv^2$, where D is a square free positive integer. Consequently,

$$4p = u^2 + Dv^2 \tag{6}$$

for some integer u satisfying

$$m = p + 1 \pm u \tag{7}$$

D is called a *CM discriminant for the prime* p and the elliptic curve has a *CM by* D. The CM method uses D in order to determine the j-invariant and constructs an EC of order $p+1-u$ or $p+1+u$.

The method starts with a prime p and then chooses the smallest D along with an integer u to satisfy Eq. (6). Then, checks whether $p+1-u$ and/or $p+1+u$ is suitable. If neither is suitable, the process is repeated. Otherwise, the so-called Hilbert polynomials (see Section 4) have to be constructed (based on D) and their roots have to be found. A root of the Hilbert polynomial is the j-invariant we are seeking. The EC and its twist are then constructed as explained in Section 2. Since only one of the ECs has the required suitable order, the particular one can be found using Langrange's theorem by picking random points P in each EC until a point is found in some curve for which $mP \neq \mathcal{O}$. Then, the other curve is the one we are seeking.

A major problem of the CM method is the construction of the Hilbert polynomials which require high precision floating point and complex arithmetic that makes their computation very expensive.

To overcome this problem, a variant of the CM method was proposed in [16]. It takes as input a CM discriminant D ($D \equiv 3 \pmod{4}$), and subsequently calculates p and m, where the only condition posed on m is that it should be a prime. The prime p is found by first picking randomly u and v of appropriate sizes, and then checking if $(u^2 + Dv^2)/4$ is prime. An important aspect of the variant concerns the computation of the Hilbert polynomials: since they depend only on D (and not on p), they can be constructed in a preprocessing phase and stored for later use. Hence, the burden of their construction is excluded from the generation of the EC.

In the rest of the section, we shall describe an alternative to the variant of [16] with which some similarities are shared: our variant takes also as input a CM discriminant D, and then computes p and m. The differences are that it uses Weber instead of Hilbert polynomials, selects p at random (or selects it from a set of prescribed primes), computes u and v in a different way (using Cornacchia's algorithm [6]), and requires m to be suitable (cf. Section 2). Actually, the order m of the elliptic curves that we generate is of the form $m = nq$, where n is a small integer and q is a large prime (larger than 2^{160}). Weber polynomials is the default choice of our variant, since they require much less precision and, as our experiments show, result in much more efficient computation of ECs. (We would like to mention that Hilbert polynomials can be equally used as well.) The polynomials, like in [16], are also constructed in a preprocessing phase.

In the following, we shall give the main steps of our variant. In order to facilitate the discussion of the experiments in Section 6, we will include also the choice of Hilbert polynomials in the description.

Preprocessing Phase.

1. Choose a discriminant $D \equiv 0$ or $3 \pmod 4$ and $D \not\equiv 3 \pmod 8$. In the following section we will explain why this limitation is necessary.
2. Construct the Weber (or the Hilbert) polynomial using the discriminant D.

Main Phase.

3. Produce randomly (or select) a prime p and check whether Eq. (6) has a solution (u, v), where u, v are integers, using Cornacchia's algorithm [6]. This algorithm solves a slightly different form, namely the equation $p = x^2 + dy^2$, but it is easy to convert Eq. (6) into this form. If a solution (u, v) is found, then proceed with the next step. Otherwise, another prime p is chosen and the step is repeated. The prime number p is going to be the order of the underlying finite field F_p.
4. Having found a solution (u, v), the possible orders of the elliptic curve are $m = p + 1 - u$ and $m = p + 1 + u$. Check if (at least) one of them is suitable. If none is suitable, then return to Step 3. Otherwise, m is the order of the elliptic curve that we will generate and proceed to the next step.
5. Compute the roots (modulo p) of the Weber (or Hilbert) polynomial. This is accomplished by using a slight modification of Berlekamp's algorithm [2]. Transform the roots of the Weber polynomial (if it has been chosen) to the roots of the corresponding Hilbert polynomial (constructed using the same D).
6. Each (Hilbert) root computed in Step 5 represents a j-invariant. Construct the two ECs as described in Section 2 (cf. Eq. (1) and (5)).
7. Determine which one of the two ECs is of a suitable order: repeatedly pick random points P on each elliptic curve, until a point is found for which $mP \neq \mathcal{O}$. Then, we are certain that the other curve is the one we seek.

The most complicated part of the CM method is the construction of the polynomials (Weber or Hilbert). This construction is presented in the following section.

4 Construction of Hilbert and Weber Polynomials

In this section we shall elaborate more on the Hilbert and Weber polynomials and discuss their strengths and limitations.

The CM discriminant D is the only input in the construction of Hilbert and Weber polynomials, denoted by $H_D(x)$ and $W_D(x)$ respectively. They both require complex and floating point arithmetic. The drawback of Hilbert polynomials is that their coefficients can be huge and their construction demands high precision. This implies that their construction can be very time consuming and possibly impossible to be implemented in systems of limited memory or with time constraints. Weber polynomials on the other hand, have much smaller coefficients and therefore the precision that is needed for their construction is not very high. In our code, we have implemented both polynomials and in the following sections we present a comparison between them.

The Hilbert polynomial $H_D(x)$, for a given positive value of D, is defined as

$$H_D(x) = \prod_{\tau} (x - j(\tau)) \tag{8}$$

for a set of values of τ given by the expression $\tau = (-\beta + \sqrt{-D})/2\alpha$, for all integers α, β, and γ that satisfy the conditions: (i) $\beta^2 - 4\alpha\gamma = -D$, (ii) $|\beta| \leq \alpha \leq \sqrt{D/3}$, and (iii) $\alpha \leq \gamma$, (iv) $\gcd(\alpha, \beta, \gamma) = 1$, and (v) if $|\beta| = \alpha$ or $\alpha = \gamma$, then $\beta \geq 0$. We shall write $H_D[j](x)$ for $H_D(x)$ when we want to emphasize the class invariant $j(\tau)$ in the construction of the polynomial.

Note that the pairs (α, β) that satisfy the above conditions are finite, which in turn implies that the values of τ are finite and consequently the factors in the Hilbert polynomial in Eq. (8). Let

$$z = e^{2\pi\sqrt{-1}\tau} \quad \text{and} \quad h(\tau) = \frac{\Delta(2\tau)}{\Delta(\tau)} \tag{9}$$

where

$$\Delta(\tau) = z \left(1 + \sum_{n \geq 1} (-1)^n \left(z^{n(3n-1)/2} + z^{n(3n+1)/2} \right) \right)^{24} . \tag{10}$$

Then, the term $j(\tau)$ (the class invariant) is defined as

$$j(\tau) = \frac{(256h(\tau) + 1)^3}{h(\tau)}. \tag{11}$$

The method we followed for the construction of the Hilbert polynomials is the one described in [4]. Let h be the *degree* or *class number* of $H_D(x)$. The bit-precision required for the generation of the Hilbert polynomials (see [1,4]) is

$$\text{H-Prec}(D) = v_0 + \binom{h}{\lfloor h/2 \rfloor} \frac{\pi\sqrt{D}}{\ln 2} \sum_{\tau} \frac{1}{\alpha} \tag{12}$$

where the sum runs over the same values of τ as the product in Eq. (8) and v_0 is a positive constant that takes care of rounding errors (typically $v_0 = 33$). Clearly, H-Prec(D) can be rather high.

The above prohibitively large precision required for the computation of the coefficients of Hilbert polynomials (even for moderate values of D) can be circumvented by using the Weber polynomials which required much smaller precision for the computation of their coefficients.

To define the Weber polynomial $W_D(x)$ we follow the approaches in [1,8]. Let α, β, γ be integers that satisfy the conditions: (i) $\beta^2 - \alpha\gamma = -D$, (ii) $|2\beta| \leq \alpha \leq \gamma$, (iii) $\gcd(\alpha, 2\beta, \gamma) = 1$, and (iv) if $2|\beta| = \alpha$ or $\alpha = \gamma$, then $\beta \geq 0$. Additionally, let $\theta = z^{-1}$ and $F(z) = (\varDelta(\tau)/z)^{1/24}$ (where the complex number z and the function $\varDelta(\tau)$ are those defined in Eq. (9) and (10)).

The construction of the polynomials is based on the so-called Weber functions which are defined as follows:

$$\begin{aligned} f_0(\alpha,\beta,\gamma) &= \theta^{-1/24} F(-\theta)/F(\theta^2) \\ f_1(\alpha,\beta,\gamma) &= \theta^{-1/24} F(\theta)/F(\theta^2) \\ f_2(\alpha,\beta,\gamma) &= \sqrt{2}\ \ \theta^{1/12} F(\theta^4)/F(\theta^2) \end{aligned}$$

For ease of notation, we shall occasionally drop in the following the arguments α, β, γ from the Weber functions. Let

$$\gamma_3 = (f_0^{24} + 8)(f_1^8 - f_2^8)/f_0^8$$

Then, given D, the Weber polynomials are defined as follows:

1. If $D \not\equiv 0 \pmod 3$ and $D \not\equiv 3 \pmod 8$, then
   a) If $D \equiv 7 \pmod 8$, then $W_D(x) = H_{4D}[f_0/\sqrt{2}](x)$
   b) If $D/4 \equiv 2$ or $6 \pmod 8$, then $W_D(x) = H_D[f_1/\sqrt{2}](x)$
   c) If $D/4 \equiv 5 \pmod 8$, then $W_D(x) = H_D[f_0^4](x)$
   d) If $D/4 \equiv 1 \pmod 8$, then $W_D(x) = H_D[f_0^2/\sqrt{2}](x)$
2. If $D \equiv 3 \pmod 6$, then $W_D(x) = H_D[\sqrt{-D}\gamma_3](x)$
3. Otherwise, $W_D(x) = H_D(x)$.

The above mathematical definition can be alternatively summarized by the following equation given in [8], which actually helped us in the implementation to easily construct the polynomials:

$$W_D(x) = \prod_i (x - \mathcal{C}(\alpha_i, \beta_i, \gamma_i)) \tag{13}$$

where $\alpha_i, \beta_i, \gamma_i$ satisfy the above mentioned conditions for α, β, γ, the values of i run over all possible reduced symmetric matrices $\left(\begin{smallmatrix}\alpha_i & \beta_i \\ \beta_i & \gamma_i\end{smallmatrix}\right)$ which have $D = \alpha_i\gamma_i - \beta_i^2$ as a positive square-free determinant, and the function $\mathcal{C}$ is defined as

$$\mathcal{C}(\alpha_i, \beta_i, \gamma_i) = \left[N \exp\left(\frac{-\pi\sqrt{-1}KBL}{24} \right) 2^{-I/6} \left(f_J(\alpha_i, \beta_i, \gamma_i) \right)^K \right]^G$$

where $J \in \{0, 1, 2\}$, $G = \gcd(D, 3)$, $I, K \in [0, 6]$, and L, N are positive integers. The precise values of these parameters depend on certain, rather tedious, conditions among α, γ and D that encompass the various cases of the mathematical definition of the Weber polynomials; the interested reader can find all the details in [8].

The bit-precision required for the construction of the Weber polynomials (see e.g., [23]) is

$$\text{W-Prec}(D) = v_0 + \frac{\pi\sqrt{D}}{\ln 2} \sum_i \frac{1}{\alpha_i} \tag{14}$$

where the sum runs over the same values of i as the product in Eq. (13). Hence, the precision for the construction of the two polynomials differ by a multiplicative factor of $\binom{h}{\lfloor h/2 \rfloor}$. This factor increases as the degree of the polynomials increases. Our experimental results confirm this fact and demonstrate the difference in precision and time efficiency between the construction of Hilbert and Weber polynomials.

To get an idea on the size of coefficients of Hilbert and Weber polynomials as well as on their space requirements for storing them off-line, we next give three examples for different values of D.

$$\begin{aligned} W_{40}(x) &= x^2 - x - 1 \\ H_{40}(x) &= x^2 - 425692800x + 910314547200 \end{aligned}$$

$$\begin{aligned} W_{292}(x) &= x^4 - 5x^3 - 10x^2 - 5x + 1 \\ H_{292}(x) &= x^4 \\ &\quad -20628770986042830460800x^3 \\ &\quad -93693622511929038759497066112000000x^2 \\ &\quad +45521551386379385369629968384000000000x \\ &\quad -380259461042512404779990642688000000000000 \end{aligned}$$

$$\begin{aligned}
W_{472}(x) &= x^6 - 12x^5 - 22x^3 - 12x - 1 \\
H_{472}(x) &= x^6 - 4383708609383203692786685920000x^5 \\
&\quad +2902435100381599559257269068222097663360000000x^4 \\
&\quad -6621978932864958986465185964976874629120000000000x^3 \\
&\quad +896632690216502725937652246573453867048960000000000000x^2 \\
&\quad +77827628475557924086643717208566407495680000000000000000x \\
&\quad +(8476837240896000000)^3
\end{aligned}$$

It is clear that the memory required for the storage of Hilbert polynomials is considerably larger than that required by the Weber polynomials.

Table 1. Transforming a root R_W of a Weber polynomial to a root R_H of the corresponding Hilbert polynomial.

d mod 8	R_H
1	$\frac{(64R_W^{12}-16)^3}{64R_W^{12}} \mod p$
2 or 6	$\frac{(64R_W^{12}+16)^3}{64R_W^{12}} \mod p$
5	$\frac{(64R_W^{6}-16)^3}{64R_W^{6}} \mod p$
7	$\frac{(R_W^{-24}-16)^3}{R_W^{-24}} \mod p$

We argued that the use of the Weber polynomials has many advantages compared to the Hilbert polynomials. However, if we choose to use them in the CM method, we must transform their roots to the roots of the corresponding Hilbert polynomials. To accomplish this, the two polynomials must have the same degree in order to associate one root of the Weber polynomial to one of the Hilbert polynomial. For discriminant $D \equiv 3 \pmod 8$ the Weber polynomial's degree is two times the degree of the corresponding Hilbert polynomial and this is the reason why we discard those values of D. A detailed analysis of transforming a root R_W of a Weber polynomial to its corresponding root R_H of a Hilbert polynomial is presented in [23]. The analysis results in a table that summarizes the transformation, a modified version of which (due to a different polynomial representation we use) is presented in Table 1. The value of d is determined as follows. If $D \equiv 0 \pmod 4$, then $d = D/4$, otherwise, $d = D$.

5 Implementation

In this section, we will discuss some issues regarding the implementation of our variant of the Complex Multiplication method. The implementation has been entirely written in ANSI C using the GNU Multiple Precision [7] library for high precision floating point arithmetic and also for the generating and manipulating integers of unlimited precision. Our implementation is also part of a software library for EC cryptography that we build [11]. The library is available from `http://www.ceid.upatras.gr/faculty/zaro/software/ecc-lib/`.

The GNUMP library, uses as a basic precision unit the *limb*, which is composed of 32 bits. Every floating point number in this library is represented by an integral number of limbs. One may modify the precision with which the floating operations are carried out using a special function that changes the number of limbs. Note, however, that 2 limbs are the minimum precision required by GNUMP for any computation.

As a first step, we implemented the basic algebraic operations for elliptic curve arithmetic. We then turned our attention to the most demanding step of the CM method, which was the construction of the Hilbert and Weber polynomials. They both require high-precision complex and floating point arithmetic with the greater demands placed, of course, by Hilbert polynomials. Also, the operations involved required the implementation of functions such as $\cos(x)$, $\sin(x)$, $\exp(x)$, $\ln(x)$, $\arctan(x)$ and $\sqrt{x}$. Since the basic complex number algebraic operations (addition, multiplication, exponentiation, and squaring) as well as a high precision floating point implementation of the above functions did not exist in GNUMP, we had to implement them from scratch. For the implementation of the particular functions we used their Taylor series expansion. As a starting point for the construction of the Hilbert polynomials, we used the code given in [24] which we considerably modified in order to support high precision floating point arithmetic. For the construction of the Weber polynomials we implemented the functions described in the IEEE Standard P1363 [8], adopting a slightly different way for producing the coefficients α, β, γ described in the standard. For the computation of the roots of polynomials modulo a prime, we used the code given in [24], which we had to modify in order to handle correctly prime numbers of any precision. Finally, the test for the suitability of the order m was done as follows. The order must be of the form $m = nq$, where n is an integer and q is a large prime (greater than 2^{160}). The test proceeds by factoring m and demanding that there are at most four small factors (smaller than 20), while one factor should be prime. If this fails, then the particular m is rejected and the process is repeated. It is easy to see that in this way, q is greater than 2^{160} for sizes of 192 or 224 bits for the field's order, since n is at most 20^4.

6 Experimental Results

Our experiments were carried out on a Pentium III (933 MHz) with 256 MB of main memory, running SuSE-Linux 7.1, and using the ANSI C gcc-2.95.2 compiler (along with the GNUMP library). All reported times are averages over 200 ECs per value of the discriminant D. For the size of the field's order, we considered two values, namely 192 and 224 bits. Our code has size 69KB, including the code for the generation of the polynomials (exclusion of the latter reduces the code size to 56KB).

We first considered experiments regarding the construction of Hilbert and Weber polynomials. Table 2 illustrates, for various values of D and h (degree of polynomial), the required limb-precision, the number of Taylor terms, and the total time for the construction. As it turns out, the construction of Weber polynomials can be done incredibly faster, and requires a much smaller number of Taylor expansion terms. In addition, it requires only 2 limbs of precision (i.e., the minimum in terms of GNUMP) for all cases considered, while the construction of the Hilbert polynomials requires from 2 to 7 limbs depending on the values of D and h (we noticed that 2 limbs were sufficient for Weber polynomials even for larger values of D and h, e.g., $D = 9640$ and $h = 16$). Note also that the precision required by the Hilbert polynomials increases with D even if h remains the same. An interesting observation concerns the cases marked with an asterisk in Table 2.

Table 2. Construction of Weber and Hilbert polynomials. (*) Coefficients of Hilbert polynomials do not have trailing decimal zeros.

		Weber polynomial			Hilbert polynomial		
D	h	limb-precision	Taylor terms	Time	limb-precision	Taylor terms	Time
20	2	2	6	0.07	2	16	0.59
40	2	2	7	0.09	2	19	0.75
52	2	2	12	0.16	2	24	1.11
88	2	2	13	0.19	3	26	1.38
148	2	2	21	0.40	3	36	2.48
232	2	2	24	0.49	4	39	3.37
39*	4	2	20	0.54	3	32	3.29
56*	4	2	10	0.20	3	63	13.61
68	4	2	11	0.23	3	72	10.84
84*	4	2	18	0.61	3	175	237.82
120	4	2	20	0.70	4	39	7.06
132	4	2	21	0.83	4	38	6.50
136*	4	2	18	0.45	4	130	72.90
168	4	2	23	0.94	4	44	8.82
184*	4	2	22	0.64	4	197	263.55
228	4	2	27	1.26	5	51	13.01
292	4	2	28	1.00	5	90	37.97
116*	6	2	19	0.67	4	190	346.57
152	6	2	20	0.73	5	149	182.24
244*	6	2	27	1.26	5	329	1493.32
472	6	2	36	2.20	7	485	4980.67

The coefficients of the corresponding Hilbert polynomials in these cases do not have trailing decimal zeros and this seems to require a higher number of Taylor terms in order for the computations to converge. We observed that this situation does not occur when D is even and ends in 0, 2 and 8. Since the trailing zeros can be stored in a compact way, this observation would suggest which Hilbert polynomials to consider for off-line computation and storage (if one wishes to use them).

A final remark concerns the comparison of the theoretically required precision, according to Eq. (12) and (14), with that measured experimentally. Our experiments have shown that a smaller precision is required in practice. For example, for $D \in \{232, 292, 472\}$, the equations give for the Hilbert polynomials bit-precisions of H-Prec(232) = 364, H-Prec(292) = 1166, H-Prec(472) = 4983, and for the Weber polynomials bit-precisions W-Prec(232) = 215, W-Prec(292) = 249, W-Prec(472) = 312. Clearly, the precisions given in Table 2 (as multiples of 32 bits) are much smaller than these numbers.

We next turn to the efficiency of our CM implementation using only Weber polynomials. Let $\#p$ denote the number of primes that we had to try in order to find a solution (u, v) using Cornacchia's algorithm (Step 3), and let $\#m$ be the number of orders m that we tried until a suitable one was found. We shall denote by $T(p, m)$ the time required to find a prime p and a suitable order m (Steps 3 and 4), by T_5 the time required for the computation of roots of the polynomial modulo p (Step 5), by T_{67} the time required for the construction of the elliptic curve (Steps 6 and 7), and by T_{main} the total time of the main phase (Steps 3-7) of our variant. The Weber polynomials have been constructed off-line during the preprocessing phase.

Table 3. Timing estimations (in secs) of our CM variant in the 192-bit finite field.

D	h	$\#p$	$\#m$	$T(p, m)$	T_5	T_{67}	T_{main}
232	2	4	5	0.63	0.01	0.32	0.96
568	4	7	6	1.02	0.04	0.33	1.39
1432	6	12	5	1.27	0.09	0.33	1.69
3448	8	15	5	1.34	0.14	0.35	1.83
5272	10	21	5	2.04	0.21	0.38	2.63
8248	12	24	5	2.39	0.32	0.31	3.02
9172	14	28	5	2.80	0.41	0.33	3.54
9640	16	33	6	3.69	0.51	0.39	4.59
9832	18	37	7	4.55	0.76	0.35	5.66
19492	20	42	5	4.78	1.22	0.30	6.30
29908	30	59	5	6.51	1.77	0.40	8.68
39796	50	102	6	11.73	6.11	0.39	18.23
39608	100	195	8	27.42	23.45	0.35	51.22

Table 3 reports the values of the above parameters for various values of D and h and shows where exactly the time is spent throughout the steps of our CM variant. According to [4], we have to try roughly $2h$ primes before a solution can be found by Cornacchia's algorithm. This fact was verified by our experiments with surprising accuracy (cf. the third column of Table 3). The number of trials for order m are approximately the same regardless of the degree of the polynomial, which is reasonable as m is directly associated with the prime p which we choose at random. Therefore, we do not expect that the number of trials required will increase as the discriminant D increases. As expected, all

Table 4. Timing estimations (in secs) of our CM variant.

			192 bits				224 bits			
D	h	$T[W]$	$\#p$	$\#m$	$T(p,m)$	T_{main}	$\#p$	$\#m$	$T(p,m)$	T_{main}
20	2	0.07	4	10	0.82	1.23	4	10	1.30	1.72
40	2	0.09	3	9	0.73	1.12	4	9	1.21	1.60
52	2	0.16	4	8	0.75	1.02	4	8	1.12	1.53
88	2	0.19	4	7	0.61	0.94	4	8	0.98	1.41
232	2	0.49	4	5	0.63	0.96	4	6	0.83	1.30
56	4	0.20	7	10	1.45	1.88	7	11	2.63	3.20
84	4	0.61	7	9	1.40	1.77	8	9	2.29	2.83
136	4	0.45	8	8	1.43	1.80	7	9	2.27	2.77
292	4	1.00	7	6	1.13	1.45	8	8	2.20	2.68
568	4	2.00	7	6	1.02	1.39	8	7	1.98	2.43
116	6	0.67	11	9	1.89	2.33	11	13	3.90	4.39
244	6	1.26	11	9	1.86	2.30	12	11	3.58	4.17
472	6	2.20	11	8	1.52	1.96	12	10	3.33	3.96
1048	6	4.80	13	6	1.45	1.90	11	8	2.92	3.55
1432	6	6.64	12	5	1.27	1.69	11	6	2.03	2.57
376	8	2.26	16	8	2.34	2.79	16	10	4.41	4.95
952	8	6.34	16	7	2.22	2.75	15	9	3.74	4.38
1528	8	9.44	16	7	2.19	2.64	16	6	3.49	4.04
2212	8	16.99	16	6	1.71	2.18	16	6	2.94	3.20
3448	8	23.66	15	5	1.34	1.83	17	5	2.65	3.01
296	10	2.00	20	10	3.70	4.23	19	13	6.58	7.23
724	10	5.33	20	9	3.44	4.07	20	12	6.51	7.19
1268	10	9.80	20	6	2.29	2.85	19	9	4.28	5.01
3412	10	29.17	20	6	2.20	2.76	20	7	3.68	4.37
5272	10	46.49	21	5	2.04	2.63	20	5	2.84	3.54

times (except for T_{67}) increase as the degree h of the polynomial increases. The most time consuming step, as D and h increase, is the computation of the roots of the polynomials.

Table 4 elaborates further by reporting values for the most important parameters regarding various values of D for the same value of h. In the table, $T[W]$ denotes the time for constructing the Weber polynomial. A first observation is that both $T(p,m)$ and T_{main} decrease as D increases, while h remains the same. Another interesting observation is that for reasonably small values of h (which do not necessarily compromise security), our variant remains efficient even in the case where it is required that the computation of Weber polynomials should be made on-line (e.g., due to limited resources posed by hardware devices).

Comparison with related work. The implementation of the CM variant in [16] was done in `C++` using the NTL library [19], which is a high-performance `C++` library for number theory and polynomial arithmetic. Also, their implementation was equipped with clever heuristics to find quickly p and m. Their experiments were done on a Pentium PC (450 MHz) running Windows NT, considering the same sizes of p (192 and 224 bits) and roughly similar values of D and h as we use. The size of their code was 164KB, excluding the code for precomputing the Hilbert polynomials which was done with MAPLE. In our implementation we didn't use any kind of heuristics. On the positive side, our variant uses considerably fewer iterations to find a suitable m, and is faster[1] compared to the times reported in [16] for (at least) all $h \leq 30$. On the negative side, the construction time of our variant degrades when h increases above 30 and D is sufficiently large. This is due to two reasons: (a) The efficient heuristics used in [16] to find p result in a number of iterations proportional to $ch/\sqrt{D}$ (for some constant $c \approx 300$), while in our variant the number of iterations is roughly $2h$. Hence, the larger the D, the less iterations are made by the variant of [16]. Moreover, our checking of the suitability conditions for m take clearly more time than simply checking on whether m is prime. (b) Our implementation takes more time to find the roots of the Weber polynomial than the time required by the corresponding function of the NTL library. We plan to further investigate the latter issue, as it is clear from Table 3 that it will considerably improve the total time.

There are two other efficient `C++` implementations of the CM method [3,18]. The latter uses the MIRACL [13] library and requires more code space (204KB) than that in [16]. The former uses the advanced `C++` library LiDIA [10] whose adaptation to embedded systems seems very difficult (if at all possible).

7 Conclusions

We have presented an implementation of a variant of the Complex Multiplication method for generating secure ECs. The variant uses Weber polynomials which can be either precomputed off-line and stored as their storage requirements are very low, or (if there are space limitations) can be constructed on-line without sacrificing efficiency (at least for small values of h).

Acknowledgments. We would like to thank the referees for their helpful comments.

References

1. A. O. L. Atkin, F. Morain, Elliptic curves and primality proving, *Mathematics of Computation* 61(1993), pp. 29–67.
2. E. R. Berlekamp, Factoring polynomials over large finite fields, *Mathematics of Computation* 24(1970), pp. 713–735.
3. H. Baier, and J. Buchmann, Efficient construction of cryptographically strong elliptic curves, in *Progress in Cryptology* – INDOCRYPT 2000, Lecture Notes in Computer Science Vol. 1977 (Springer-Verlag, 2000), pp. 191–202.

[1] The times given in our tables should be roughly doubled in order to be compared with the times reported in [16].

4. Ian Blake, Gadiel Seroussi, and Nigel Smart, *Elliptic curves in cryptography* , London Mathematical Society Lecture Note Series 265, Cambridge University Press, 1999.
5. D. Burton, *Elementary Number Theory*, McGraw Hill, 4th edition, 1998.
6. G. Cornacchia, Su di un metodo per la risoluzione in numeri interi dell' equazione $\sum_{h=0}^{n} C_h x^{n-h} y^h = P$, *Giornale di Matematiche di Battaglini* 46 (1908), pp. 33–90.
7. GNU multiple precision library, edition 3.1.1, September 2000. Available at: `http://www.swox.com/gmp`.
8. IEEE P1363/D13, *Standard Specifications for Public-Key Cryptography*, ballot draft, 1999. `http://grouper.ieee.org/groups/1363/tradPK/draft.html`.
9. Implementations of Portions of the P1363 Draft. `http://grouper.ieee.org/groups/1363/P1363/implementations.html`.
10. LiDIA. *A library for computational number theory*, Technical University of Darmstadt. Available from `http://www.informatik.tu-darmstadt.de/TI/LiDIA/Welcome.html`.
11. E. Konstantinou, Y. Stamatiou, and C. Zaroliagis, A Software Library for Elliptic Curve Cryptography, in *Proc. 10th European Symposium on Algorithms* – ESA 2002 (Engineering and Applications Track), Lecture Notes in Computer Science (Springer-Verlag, 2002), to appear.
12. A. J. Menezes, T. Okamoto and S. A. Vanstone, Reducing elliptic curve logarithms to a finite field, *IEEE Trans. Info. Theory*, 39(1993), pp. 1639–1646.
13. Multiprecision Integer and Rational Arithmetic C/C++ Library, `http://indigo.ie/~mscott/`.
14. G. C. Pohlig and M. E. Hellman, An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance, *IEEE Trans. Info. Theory*, 24 (1978), pp. 106–110.
15. T. Satoh and K. Araki, Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curves, *Comm. Math. Univ. Sancti Pauli*, 47(1998), pp. 81–91.
16. Erkay Savas, Thomas A. Schmidt, and Cetin K. Koc, Generating Elliptic Curves of Prime Order, in *Cryptographic Hardware and Embedded Systems* – CHES 2001, Lecture Notes in Computer Science Vol. 2162 (Springer-Verlag, 2001), pp. 145–161.
17. R. Schoof, Counting points on elliptic curves over finite fields, *J. Theorie des Nombres de Bordeaux*, 7(1995), pp. 219–254.
18. M. Scott, A C++ Implementation of the Complex Multiplication (CM) Elliptic Curve Generation Algorithm from Annex A, in *Implementations of Portions of the P1363 Draft*. `http://grouper.ieee.org/groups/1363/P1363/implementations.html`.
19. V. Shoup, NTL: A Library for doing Number Theory, URL: `http://shoup.net/ntl/`.
20. I. A. Semaev, Evaluation of discrete logarithms on some elliptic curves, *Mathematics of Computation*, 67(1998), pp. 353–356.
21. J. H. Silverman, *The Arithmetic of Elliptic Curves*, Springer-Verlag, GTM 106, 1986.
22. N. P. Smart, The discrete logarithm problem on elliptic curves of trace one, *Journal of Cryptography*, 12(1999), pp. 193–196.
23. Thomas Valente, *A distributed approach to proving large numbers prime*, Rensselaer Polytechnic Institute Troy, New York, Thesis, August 1992.
24. Pate Williams. Available at: `http://www.mindspring.com/~pate`.