A Software Library for Elliptic Curve Cryptography*

Elisavet Konstantinou[1,2], Yiannis 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 an implementation of an EC cryptographic library targeting three main objectives: portability, modularity, and ease of use. Our goal is to provide a fully-equipped library of portable source code with clearly separated modules that allows for easy development of EC cryptographic protocols, and which can be readily tailored to suit different requirements and user needs. We discuss several implementation issues regarding the development of the library and report on some preliminary experiments.

1 Introduction

Cryptographic systems based on elliptic curves were introduced independently by Koblitz [15] and Miller [20] in 1985 as an alternative to conventional public key cryptosystems such as RSA [23] and DSA [14]. The main advantage of elliptic curve (EC) cryptosystems is that they use smaller parameters (e.g., encryption key) than the conventional cryptosystems. The reason is that the underlying mathematical problem on which their security is based, the EC discrete logarithm problem (ECDLP), appears to require more time to solve than the analogous problem in groups generated by prime numbers on which the conventional cryptosystems are based. For groups defined on ECs (where the group elements are the points of the EC), the best algorithm for attacking this problem takes time exponential in the size of the group, while for groups generated by prime numbers there are algorithms that take subexponential time. This implies that one may use smaller parameters for the EC cryptosystems than the parameters used in RSA or DSA, obtaining the same level of security. A typical example is that a 160-bit key of an EC cryptosystem is equivalent to RSA and DSA with a 1024-bit modulus. As a consequence, smaller keys result in faster implementations, less storage space, as well as reduced processing and bandwidth requirements.

* 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).

R. Möhring and R. Raman (Eds.): ESA 2002, LNCS 2461, pp. 625–637, 2002.
© Springer-Verlag Berlin Heidelberg 2002

There are many important decisions that one should make before starting implementing an elliptic curve cryptosystem (ECC). These include the type of the underlying finite field, the algorithms for implementing the basic algebraic operations, the type of the elliptic curve to be used as well as its generation, and finally the elliptic curve protocols. The fields usually used are either prime fields (denoted by F_p, where p is a prime) or binary fields. Selecting a prime field (which will be our concern here) implies proper choice of p, since all basic operations will be modulo that prime and the security of the system will depend on its size. The larger the prime, the more secure but slower the cryptosystem. The basic algebraic operations of the EC group that must be implemented are addition of points on an EC, and scalar multiplication (multiplication of a point by an integer). The latter is the operation upon which the ECDLP is based. There should also be a way for generating secure elliptic curves and create random points in them. The generation of such curves can be accomplished with three methods, namely the point counting method (see [3]), the method based on the constructive Weil descent [8], and the Complex Multiplication method (see [3]). The latter two methods build ECs of a *suitable order* (where order refers to the number of elements in the group defined by the EC), i.e., the order satisfies certain conditions, necessary for the cryptographic strength of the EC. The former method does not necessarily produce ECs of a suitable order. Once all these basic operations have been implemented, one can start developing cryptographic protocols.

Several studies have been conducted in the past and many papers have been written on software implementations of ECCs. The majority of them focuses either on the efficient implementation of the basic algebraic operations (see e.g., the excellent surveys of [18] and [19]), or on the efficient implementation of a single protocol for one particular finite field. For example, in [4] a nice study of the performance of the Elliptic Curve Discrete Signature Algorithm (ECDSA) over prime fields is presented using the NIST recommended ECs, while in [10] a similar study is presented regarding binary fields.

In this paper, we present an implementation of an EC cryptographic library targeting three main objectives: portability, modularity, and ease of use. Our goal is to provide a fully-equipped library of portable source code with clearly separated modules which allow for the easy development of EC cryptographic protocols, and which can be readily tailored to suit different requirements and user needs. The library has been implemented in ANSI C using the GNUMP [9] library. We have chosen prime fields as the underlying field of our implementations, because of their simplicity in representation and in performing algebraic operations. Our library includes all the basic operations of an EC group, several cryptographic protocols based on ECs (including the EC Discrete Signature Algorithm – ECDSA), and the Complex Multiplication (CM) method for generating secure elliptic curves. The implementation of the latter method included several engineering challenges due to the fact that it heavily relies on the ability to perform unlimited precision computations on complex numbers, and in addition requires the use of special trigonometric and exponentiation functions.

The reason for high precision stems from a crucial step of the method, namely the construction of the so-called Hilbert or Weber polynomials. The GNUMP library for high-precision floating point arithmetic lacks both an implementation of the required functions as well as of high-precision complex number arithmetic, and hence we had to implement them from scratch. For the implementation of the functions, we had to resort to their Taylor series expansion and study the interrelationship between the precision of basic complex number (floating point) arithmetic and the number of terms in the Taylor series necessary to produce correct results. Clearly, the number of terms depended on the required precision. Another problem was that the roots of the Hilbert polynomials are absolutely necessary in order to find the EC, but their construction is a considerably high burden, because of the very high precision they require. On the other hand, the construction of Weber polynomials does not require such a high precision and consequently turns out to be incredibly faster (cf. Section 5), but their roots are not appropriate for the CM method. Hence, we had to find a practical way to transform Weber roots to the roots of the corresponding Hilbert polynomial (something that was not adequately addressed in the bibliography).

At the current stage of the library, we are making no claims that the implemented algorithms are the best or fastest possible (they certainly are not, although preliminary experiments are rather encouraging). We hope, however, that our work will be valuable to those interested in developing easily protocols based on elliptic curve cryptography by providing, in a nutshell, basic and advanced cryptographic primitives and by uncovering many of the pitfalls and their treatment, inherent in their implementation. Our library is publicly available from `http://www.ceid.upatras.gr/faculty/zaro/software/ecc-lib/`.

To the best of our knowledge, there are two other libraries which offer primitives for EC cryptography: LiDIA [16] and MIRACL [21]. Both are efficient, general purpose, number-theoretic libraries implemented in `C++` that offer a vast number of cryptographic primitives. Although there seem to exist implementations of EC cryptographic protocols (e.g., ECDSA) and EC generation methods (CM method) based on these libraries, these implementations are either not publicly offered (LiDIA), or are partly provided (MIRACL).

2 Basic Concepts of Elliptic Curve Algebra

In this section we review some basic concepts regarding elliptic curves and their definition over finite fields. The interested reader may find additional informations, for example, in [3, 25]. We also assume some familiarity with elementary number theory (see e.g., [5]).

The elliptic curves are usually defined over *binary fields* F_{2^m} $(m \geq 1)$, or over *prime fields* F_p, $p > 3$. The case $p = 3$ for prime fields requires separate attention but it is, generally, not difficult to transfer results obtained for F_p, $p > 3$, to the case $p = 3$. In our library we use elliptic curves defined over prime fields.

An *elliptic curve* (EC) over the prime field F_p, denoted by $E(F_p)$, is the set of points $(x, y) \in F_p$ (represented by affine coordinates) which satisfy the equation

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

where $4a^3 + 27b^2 \neq 0$ (this condition guarantees that Eq. (1) does not have multiple roots in F_p), along with a special point denoted by $\mathcal{O}$, called the *point at infinity*. An addition operation $+$ is defined over $E(F_p)$ such that $(E(F_p), +)$ defines an Abelian group, called the *EC group*, with $\mathcal{O}$ acting as its identity. The addition operation on $E(F_p)$ is defined as: $P+Q = \mathcal{O}$, if $P = -Q$; $P+Q = Q$, if $P = \mathcal{O}$; and $P+Q = R$, otherwise, where, if $P = (x_1, y_1)$ and $Q = (x_2, y_2)$, then the coordinates of the point $R = (x_3, y_3)$ are given by $x_3 = z^2 - x_1 - x_2$, $y_3 = z(x_1 - x_3) - x_3 - y_1$ with $z = \frac{y_1 - y_2}{x_1 - x_2}$, if $P \neq Q$, and $z = \frac{3x_1^2 + a}{2y_1}$, if $P = Q$ (a is the coefficient from Eq. (1)). The negative $-Q$ of a point $Q = (x, y)$ is the point $(x, p - y)$.

A fundamental operation of cryptographic protocols based on EC is the *scalar (or point) multiplication*, i.e., the multiplication of a point P by an integer k (an operation analogous to the exponentiation in multiplicative groups), that produces another point $Q = kP$ (the point resulting by adding P to itself for k times). Several algorithms exist for the fast and efficient implementation of the scalar multiplication operation. Most of them are based on the binary representation of the integer k.

The *order m of an elliptic curve* is the number of points in $E(F_p)$. Hasse's theorem (see e.g., [3, 25]) gives upper and lower bounds for m that are based on the order p of F_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 (stating that the exponentiation of any group element to the power of the group's order gives the identity element) on $E(F_p)$, gives that $mP = \mathcal{O}$ 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.

3 Generation of Elliptic Curves and the Complex Multiplication Method

Many of the security properties of elliptic curve cryptosystems depend on the order of the EC group and this is determined by the generated EC. If this order is *suitable*, i.e., it obeys some specific good properties, then there is a guarantee for a high level of security. The order m of an EC is called *suitable*, if the following conditions are satisfied:

1. m must have a sufficiently large prime factor (greater than 2^{160}).
2. $m \neq p$.
3. For all $1 \leq k \leq 20$, it should hold that $p^k \neq 1 \bmod m$.

The above conditions ensure the robustness of cryptosystems based on the *discrete logarithm problem for EC groups* (ECDLP), since it is very difficult for all

known attacks to solve this problem efficiently, if m obeys the above properties. ECDLP asks for determining the value of t when two points P, Q in $E(F_p)$ are given such that P is of order n and $Q = tP$, where $0 \leq t \leq n-1$.

As it was mentioned in the introduction, there are three methods for generating ECs: the point counting method (see [3]), the method based on the constructive Weil descent [8], and the Complex Multiplication (CM) method (see [3]). The point counting method does not necessarily construct an EC of suitable order, but it may achieve this by repeated applications of the method. The other two methods construct ECs of a suitable order. In [8] it was shown that the method based on the constructive Weil descent suffers from a major drawback that is not easy to handle: it samples from a very small subset of the set of possible elliptic curves. For this reason, we didn't implement this particular method in our library. We only implemented the CM method and compared it to an implementation of the point counting method given in [12]. The point counting method is based on Schoof's algorithm for the exact counting of rational points on an elliptic curve (see [3] for details as well as some improvements on Schoof's algorithm). In the rest of this section we shall briefly review the CM method.

3.1 A High-Level Description of the CM Method

The Complex Multiplication method generates an EC of a suitable order and also computes the coefficients a, b which determine the EC. The method starts by creating an EC of a suitable order and then proceeds to determine the coefficients a, b. To accomplish the latter, the roots of the so-called Hilbert or Weber polynomials have to be computed. Each root of such a polynomial determines two possible elliptic curves. However, only one of them has the desired suitable order.

Let $E(F_p)$ be an elliptic curve of order m defined over the prime field F_p. Hasse's theorem (Eq. (2)) implies that the quantity $Z = 4p - (p+1-m)^2$ is positive, and that it exists a unique factorization $Z = Dv^2$, where D is a square-free positive integer and v is some integer. In addition, there exists an integer u such that:

$$4p = u^2 + Dv^2 \tag{3}$$

where u satisfies $m = p + 1 \pm u$. We say that D is a *CM discriminant* for the prime p and that $E(F_p)$ has a *CM by* D. The Complex Multiplication method takes as input some discriminant D. The basic steps of the method are as follows:

1. Pick a random prime p and check whether Eq. (3) has a solution (u, v), with u, v integers. One algorithm that can be used to solve this equation in u, v is Cornacchia's algorithm [6]. If there is no solution, then 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.
2. There are only two possible orders for the group $E(F_p)$, namely $m = p+1-u$ and $m = p + 1 + u$. Check whether (at least) one of them is suitable. If this

is the case, then proceed to Step 3 (m is the order of the elliptic curve that we will generate). Otherwise, return to Step 1.

3. Construct either the Hilbert or the Weber polynomial, using the discriminant D. It should be noted that both types of polynomials lead to the construction of the same elliptic curve.
4. Compute the roots (modulo p) of either polynomial (this is accomplished by using a slight modification of Berlekamp's algorithm [2]). To further proceed, however, the roots of the Hilbert polynomial are required. If in Step 3 the construction of the Weber polynomial was chosen, then transform their roots (cf. Subsection 3.2) to the roots of the corresponding Hilbert polynomial (constructed using the same discriminant D). From every (Hilbert) root, two elliptic curves will be generated, but only one has the desired order m. Given a root j, the first curve is given by the equation $y^2 = x^3 + ax + b$, where $a = 3k$, $b = 2k$ and $k = \frac{j}{1728-j}$ (all operations are modulo p). The second curve, called the *twist* of the first, is given by $y^2 = x^3 + ac^2x + bc^3$, where c is any quadratic non-residue in F_p.
5. Since only one of the curves has the required suitable order m, we can find the particular one using a simple procedure that is based on Langrange's theorem (for any group point P, it should hold that $mP = \mathcal{O}$): 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.

3.2 The Hilbert and Weber Polynomials

One major difficulty associated with the construction of Hilbert and Weber polynomials for a given discriminant D, denoted by $H_D(x)$ and $W_D(x)$ respectively, is the need for implementing high precision, complex arithmetic. Especially for the Hilbert polynomials, whose coefficients can become huge for large values of D, the need for high precision is more apparent. On the other hand, the Weber polynomials have smaller coefficients than the equivalent Hilbert polynomials, and their construction requires less time. Due to space limitations, we cannot provide the (rather lengthy) definitions of these polynomials, but will give an example of the two polynomials for $D = 292$ in order to get an idea on the size of the coefficients and hence on the required high precision.

$$
\begin{aligned}
W_{292}(x) &= x^4 - 5x^3 - 10x^2 - 5x + 1 \\
H_{292}(x) &= x^4 \\
&\quad -2062877098604283046080 0x^3 \\
&\quad -9369362251192903875949706611200000 0x^2 \\
&\quad +4552155138637938536962996838400000000 0x \\
&\quad -3802594610425124047799906426880000000000 0
\end{aligned}
$$

In our library, we have implemented both Hilbert and Weber polynomials, mainly in order to perform a comparative study of how their requirements for time and precision scale in connection with certain parameters such as the discriminant D.

If in Step 3 of the CM method we choose to implement the Weber polynomials, we must transform their roots into the roots of the corresponding Hilbert polynomials (generated from the same discriminant D). A detailed analysis of how this transformation can be carried out is partly presented in [26], where a different representation of the polynomials is used. By working out the details and taking into account the representation of polynomials that we use, we came up with an easy to implement transformation.

4 Implementation Considerations

In this section, we will discuss several implementation issues regarding the development of our library. Our major design goals were to develop a portable, modular, and easy-to-use library. Before designing the library components, several decisions had to be made regarding the choice of the field on which to base the library, its size, as well as the libraries for generating and performing arithmetic with large numbers (e.g., primes).

To address portability, we have chosen to implement our library in ANSI C, using the (ANSI C) GNU Multiple Precision arithmetic library [9] for integer and floating point arithmetic with infinite precision.

We have chosen prime fields F_p mainly due to their simplicity in representation and in performing the basic algebraic operations. For the representation of numbers in F_p, we use the representation of large numbers provided by GNUMP. GNUMP represents integers and floating point numbers using a number of units called *limbs*, each consisting of 32 bits (with 2 limbs being the minimum precision required for any computation). Although GNUMP supports infinite precision computations with integers, we had to enhance it with some useful integer functions such as factorization, primitive root location, etc, which were not provided[1]. We also had to augment GNUMP's floating point capabilities with implementations of the basic complex number algebraic operations (addition, multiplication, exponentiation, and squaring) as well as with a high precision floating point implementation of functions such as $\cos(x)$, $\sin(x)$, $\exp(x)$, $\ln(x)$, $\arctan(x)$ and $\sqrt{x}$ required by other modules of the library (e.g., by the one which implements the Complex Multiplication method) and which are not offered by GNUMP. We implemented these functions using their Taylor series expansion, suitably truncated for our computation needs.

To address modularity, we have built the library around a set of modules that are organized in a bottom-up fashion. There are four major modules: the *Kernel*, the *EC operations* module, the *EC generation* module, and the *Applications* module. The software modules of our final design, along with the most important components that they include and their relationships, are shown in Figure 1.

[1] We had initially used the LIP library [17] for the generation and processing of large integers mainly because it provided all these functions. However, we discovered that the LIP implementations were a source of considerable time consumption. Our implementation of these functions turned out to be much more efficient.

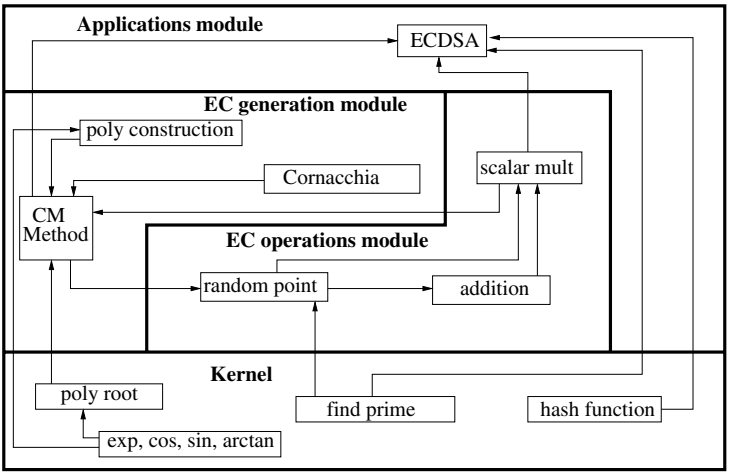

Fig. 1. The architecture of the library

The *Kernel* includes several components that perform the basic algebraic operations as well as trigonometric and exponentiation function computations using infinite precision integer and floating point arithmetic. In addition, it includes components that implement a number of more advanced (and complicated) operations, that we had to build from scratch using the primitives of the basic operations. These advanced operations include components for the manipulation of integer coefficients in polynomials (algebraic operations), and finding roots of polynomials modulo a prime number. All the components in the kernel form the core of our library and they are independent of components in other modules, at a higher level, which use them. This means that they can be optimized independently in order to tune the performance of the basic and advanced arithmetic operations.

The *EC operations* module includes several components that implement the elliptic curve related algebraic operations. We have a component in which the elliptic curve data type is defined (which is an array composed of two infinite precision numbers representing the coefficients a and b in Eq. (1)) along with the structure of a curve point (represented as a pair of two infinite precision integers). There are also components that generate random points on elliptic curves, perform addition of two points of the EC, perform scalar multiplication, and create a base point in the EC.

The *EC generation* module is one of the most important and algorithmically challenging of our library. It is composed of a number of components, including Cornacchia's algorithm [6] for solving a special class of Diophantine equations, that implement the Complex Multiplication (CM) method using both Hilbert and Weber polynomials. Two main issues had to be addressed in the implementation of the CM method: (i) the implementation of complex, high precision, floating point arithmetic as well as the use of special mathematical functions (the

handling of this issue was carried out by resorting to the Taylor series expansion of these functions as was discussed in the Kernel module); (ii) the demand for increased floating point arithmetic precision as the value of the discriminant D increases. For example, the construction of the Hilbert polynomial $H_{40}(x)$ required a precision of 2 limbs (and 19 terms in the Taylor series expansion), while the construction of the polynomial $H_{292}(x)$ required a precision of 5 limbs (and 90 terms in the Taylor series expansion). To determine the required precision, we start with some initial value and then check whether the resulting polynomial is computed correctly. If not, the precision is increased and the process is repeated.

The *Applications* module contains a number of cryptographic primitives and high level protocols based on elliptic curves including the Diffie-Hellman key exchange protocol, public/private key generation, data encryption, the Elliptic Curve Digital Signature Algorithm (ECDSA) as well as a simple one-pad encryption scheme. One may easily build a richer set of cryptographic protocols using the operations offered by the other modules of the library with only knowledge the interface with these functions (calling convention and required parameters).

The modular structure reflects the distinct nature and physical separation of the software modules of our library. It is easy to replace one method by another (perhaps more efficient), by simply implementing the new method and plugging it into our library; or one may discard, for example, the complex multiplication code and produce ECs with another method, or use a concrete, precomputed set of elliptic curves such as the ones proposed by NIST (see [7]).

5 Experiments

In this section we report on the performance of the basic components of our library based on some preliminary experiments we conducted. The experiments were carried out on a Pentium III (933 MHz) with 256 MB of main memory, using the GNU multiple precision library, and the ANSI C gcc-2.95.2 compiler. In the following, let p denote a prime, $|p|$ its size, and let $p_{|p|}$ denote a prime p of size $|p|$. We conducted experiments on three different fields with representative prime sizes, namely $F_{p_{175}}$, $F_{p_{192}}$, and $F_{p_{224}}$. Note that Hasse's theorem (Eq. (2)) as well as the first suitability condition (cf. Section 3), imply that $|p|$ must be at least 160 bits long.

We first considered the times required by the scalar multiplication operation and by the EC cryptographic protocols included in the library. We note that no attempt has been made for code optimization, or hard-coding (e.g., writing parts of the library in assembly, etc) as is customary in implementations of EC cryptographic protocols (see [18, 19]). The CPU times are shown in Table 1. As it was expected the timings increase as the size of the field increases.

We next turn to the evaluation of the CM method. To evaluate its efficiency we made an experimental comparison with the point counting method (cf. Section 3). The goal was to investigate the relationship between the time required by repeated applications of the point counting method in order to achieve an EC of suitable order, and the burden of constructing the polynomials in the CM

Table 1. Timing estimates in msecs for various modules of the library

$\|p\|$	175 bits	192 bits	224 bits
Scalar multiplication	13.6	15.7	19.5
Key generation	19.6	23.9	30.8
ECDH protocol	27.2	31.4	39
ECES encryption	28.8	36.5	46
ECES decryption	13.5	16.3	19.1
Signature	19.1	22.7	30.6
Verification	24.5	28.3	36.8

method. In all experiments, we have chosen two representative values for $|p|$, namely 175 and 192 bits.

We started with the point counting method for which we conducted several experiments. For each experiment we first generated a random prime number p, and subsequently we generated uniformly at random two integers a and b such that $1 \leq a, b \leq p$. These two numbers represent the coefficients in the equation of the elliptic curve (Eq. (1)). We then used an implementation of Schoof's algorithm from [12] to find the order of the curve. In all experiments we conducted, we observed that this is a rather time consuming method having a big variance both w.r.t. the number of repetitions to construct a suitable curve and w.r.t. to the time required. The fastest experiment took 1 repetition and 25 minutes to construct a suitable EC, while the slowest took 21 repetitions and more than 3 hours.

We next experimented with the CM method. Recall that the only input to the CM method is the discriminant D. For our experiments, we have chosen three representative values for D, namely 40, 292, and 472, that produce Hilbert and Weber polynomials of degree 2, 4, and 6, respectively. In Table 2, we report on the outcomes of our experiments for $|p| = 192$ (similar timings were reported for $|p| = 175$). It turns out that the most time consuming step concerns the construction of Hilbert or Weber polynomials (Step 3 of the CM method). In the tables, we report on the number of repetitions to find a suitable order (i.e., number of repetitions required by Steps 1 and 2), on the time required to construct the Hilbert ($T[H]$), or the Weber ($T[W]$) polynomial, and on the time required by all steps excluding $T[H]$ or $T[W]$, denoted by $T_{tot} - T[H]$ and $T_{tot} - T[W]$, respectively. All reported times are averages over 50 experiments, while the number of repetitions are maximum over all experiments.

We would like to make two comments: (1) The construction of the polynomials may seem as a drawback in a first place. However, notice that both Hilbert and Weber polynomials depend only on D (and not on p). This implies that one can generate *off-line* the polynomials for the various values of D considered, and have them handy for the generation of an EC using the CM method. This is a major advantage of this method, and hence the important times are the one appearing in the last two columns of each table. (2) Constructing Weber poly-

Table 2. The CM method for $|p| = 192$ bits. All times are in sec, unless stated otherwise

	$\|p\| = 192$ bits				
	#repetitions	$T[H]$	$T[W]$	$T_{tot} - T[H]$	$T_{tot} - T[W]$
$D = 40$	7	0.75	0.09	1.66	1.57
$D = 292$	7	37.97	1.00	2.04	1.98
$D = 472$	7	1 h 23 min 1 sec	2.20	3.19	2.56

nomials and then transform their roots to those of the corresponding Hilbert polynomial is incredibly faster than constructing directly Hilbert polynomials.

We conclude by commenting on the efficiency of our implementation of the CM method. We are aware of three other implementations. In [12], a (publicly available) implementation in `C++` is given that uses the MIRACL library [21]. The implementation follows the IEEE standard defined in [11]. It is different from ours as it takes as input a prime p and then decides D. Moreover, this implementation uses only Weber polynomials and proceeds with a different method to construct the EC, by avoiding the conversion of Weber roots to Hilbert roots. In [1] and [24] two other (non-publicly available) implementations are given which follow a method similar to ours. The former is implemented in `C++` using the LiDIA 2.0 library [16]; the latter uses the NTL library [22]. Preliminary experiments we performed with the implementation in [12] showed that our implementation was by a factor of 1.5 slower, while in some cases (e.g., $D = 40$) it was slightly faster. We believe that this is acceptable, given the fact that no effort regarding any kind of optimization has been made.

6 Concluding Remarks

We have presented the implementation of a library that supports the construction of robust elliptic curve cryptosystems based on F_p, with p a prime larger than 3. The library includes implementations of the basic algebraic operations as well as a variety of cryptographic protocols. One of the strengths of our library is that it includes an implementation of the complex multiplication method for producing elliptic curves of suitable order that ensures robustness of cryptosystems based on ECDLP. We believe that the portability and modularity of the library can be exploited in order to develop more complex protocols as well as to address efficiency issues. The later is a target which we next plan to attack.

References

[1] H. Baier, and J. Buchmann, *Efficient construction of cryptographically strong elliptic curves*, in *Progress in Cryptology* – INDOCRYPT 2000, LNCS 1977 (Springer, 2000), pp. 191-202. 635

[2] E. R. Berlekamp, *Factoring polynomials over large finite fields*, Math. Comp. 24, 111 (1970), pp. 713-735. 630

[3] Ian Blake, Gadiel Seroussi, and Nigel Smart, *Elliptic curves in cryptography* , London Math. Society Lecture Note Series 265, Cambridge University Press, 1999. 626, 627, 628, 629

[4] M. Brown, D. Hankerson, J. Lopez, and A. Menezes, Software Implementation of the NIST Elliptic Curves over Prime Fields, in *Topics in Cryptology* – CT-RSA 2001, LNCS 2020 (Springer, 2001), pp. 250-265. 626

[5] D. Burton, *Elementary Number Theory*, McGraw Hill, 4th edition, 1998. 627

[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), 33–90. 629, 632

[7] CSRC, Recommended elliptic curves for federal government use, July 1999. Available at: `http://csrc.nist.gov/csrc/fedstandards.html`. 633

[8] S. Galbraith, Limitations of constructive Weil descent, in *Public-Key Cryptography and Computational Number Theory*, (Alster and Kazimierz eds.) 2001, pp. 59-70. 626, 629

[9] GNU Multiple Precision library, ed. 3.1.1, Sept 2000, `http://www.swox.com/gmp`. 626, 631

[10] D. Hankerson, J. Lopez, and A. Menezes, Software Implementation of Elliptic Curve Cryptography over Binary Fields, in *Cryptographic Hardware and Embedded Systems* – CHES 2000, LNCS 1965 (Springer, 2000), pp. 1-24. 626

[11] IEEE P1363/D13, *Standard Specifications for Public-Key Cryptography*, ballot draft, 1999. `http://grouper.ieee.org/groups/1363/tradPK/draft.html`. 635

[12] Implementations of Portions of the P1363 Draft. `http://grouper.ieee.org/groups/1363/P1363/implementations.html`. 629, 634, 635

[13] Don Johnson, and Alfred Menezes, *The Elliptic Curve Digital Signature Algorithm (ECDSA)*, Tech. Report CORR 99-06, Dept of Combinatorics and Optimization, Univ. of Waterloo, 1999. Available at: `http://www.cacr.math.uwaterloo.ca/`.

[14] D. W. Kravitz, Digital Signature Algorithm, U. S. Patent #5,231,668, 27 July 1993. 625

[15] N. Koblitz, *Elliptic curve cryptosystems*, Math. of Comp., 48 (1987), pp.203-209. 625

[16] LiDIA. *A library for computational number theory*, Technical University of Darmstadt, Germany. Available from `http://www.informatik.tudarmstadt.de/TI/LiDIA/Welcome.html`. 627, 635

[17] LIP (Large Integer Package). Available through ftp at the directory texttt/usr/spool/ftp/pub/lenstra at server `flash.bellcore.com`. 631

[18] J. López and R. Dahab, *Performance of Elliptic Curve Cryptosystems*, Tech. Report, IC-00-08, May 2000. Available at: `http://www.dcc.unicamp.br/ic-main/publications-e.html`. 626, 633

[19] J. López and R. Dahab, *An Overview of Elliptic Curve Cryptography*, Tech. Report, IC-00-10, May 2000. Available at: `http://www.dcc.unicamp.br/ic-main/publications-e.html`. 626, 633

[20] V. Miller, Uses of elliptic curves in cryptography, in *Advances in Cryptology* – Crypto '85, LNCS 218 (Springer, 1986), pp. 417-426. 625

[21] Multiprecision Integer and Rational Arithmetic C/C++ Library, `http://indigo.ie/ mscott/`. 627, 635

[22] NTL: A Library for doing Number Theory, `http://shoup.net/ntl/`. 635

[23] R. Rivest, A. Shamir, and L. Adleman, A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, *Comm. of the ACM*, 21(1978), pp.120-126. 625

[24] Erkay Savas, Thomas A. Schmidt, and Cetin K. Koc, Generating Elliptic Curves of Prime Order, *Cryptographic Hardware and Embedded Systems* – CHES 2001, LNCS 2162 (Springer, 2001), pp. 145-161. 635

[25] J. H. Silverman, *The Arithmetic of Elliptic Curves*, Springer, GTM 106, 1986. 627, 628

[26] Thomas Valente, *A distributed approach to proving large numbers prime*, Rensselaer Polytechnic Institute Troy, New York, Thesis, August 1992. 631