Implementing the ECC Brainpool curve generation procedure using open source software

V. Gayoso Martínez and L. Hernández Encinas
Information Security Institute (ISI), Spanish National Research Council (CSIC), Madrid, Spain

Abstract—Elliptic Curve Cryptography (ECC) began to be used almost 30 years ago. Since then, ECC has been applied to an increasing number of fields (information encryption, digital signatures, integer factorization, etc.). However, one practical problem still arises when an organization decides to implement an ECC solution: what elliptic curve is the most adequate in the deployment scenario?

This contribution analyses the most important features of the elliptic curve generation procedure defined by the ECC Brainpool consortium. In addition to that, this paper describes the Java application that we have implemented following the Brainpool specifications. This application can be used for generating new elliptic curves that fulfil the security requirements defined by Brainpool. Finally, we provide the test results offered by our implementation, so interested readers can understand how much time it takes to generate elliptic curves suitable for cryptographic purposes that conform to the Brainpool specification.

Keywords: Brainpool, elliptic curves, Java, public key cryptography.

1. Introduction

In 1985, Victor Miller [1] and Neal Koblitz [2] independently proposed a cryptosystem based on elliptic curves, whose security relies on the Elliptic Curve Discrete Logarithm Problem (ECDLP). Elliptic Curve Cryptography (ECC) can be applied to data encryption and decryption, digital signatures, and key exchange procedures, among others [3], [4].

Even though elliptic curve cryptographic protocols are well defined in standards from ANSI [5], [6], IEEE [7], [8], ISO/IEC [9], NIST [10], and other similar organisations, it is usually the case that the elliptic curve parameters that are necessary to operate those protocols are offered to the reader without a complete and verifiable pseudorandom generation process. Some of the most important limitations detected across the main cryptographic standards regarding the processes for generating elliptic curves suitable for cryptography are the following [11]:

- The seeds used to generate the curve parameters are typically chosen ad hoc.
- The primes that define the underlying prime fields have a special form aimed to facilitate efficient implementations.
- The parameters specified do not cover in all the cases key lengths adapted to the security levels required nowadays.

In this scenario, a European consortium of companies and government agencies led by the Bundesamt für Sicherheit in der Informationstechnik (BSI) was formed in order to study the aforementioned limitations and produce their recommendations for a well defined elliptic curve generation procedure. The group was named ECC Brainpool (henceforth simply Brainpool) and, apart from the BSI, some of the most relevant companies and public institutions that took part were G&D, Infineon Technologies, Philips Electronics, the University of Bonn, Gemplus (now part of Gemalto), the Institute for Experimental Mathematics (University of Duisburg-Essen), Siemens, the Technical University of Darmstadt, T-Systems, Sagem Orga (now Morpho, part of the Safram group), the Institute for Applied Information Processing and Communications (Graz University of Technology), and the Secure Information Technology Center - Austria (A-SIT).

In 2005, Brainpool delivered the first version of a document entitled “ECC Brainpool standard curves and curve generation” [11], which was revised and published as a Request for Comments (RFC) memorandum in 2010, the “Elliptic Curve Cryptography (ECC) Brainpool standard curves and curve generation” [12].

This contribution analyses the elliptic curve generation procedure as defined by Brainpool in [11] and [12]. Those specifications consist of the proper introductory sections, the step by step description of the elliptic curve generation algorithm, and the validation data that can be used for checking the correctness of any software implementation.

The rest of this document is organized as follows: Section 2 presents a brief mathematical introduction to elliptic curves. Section 3 describes the most important characteristics of the Brainpool specification. Section 4 includes a functional description of GCEC, the Java application developed by us in order to generate elliptic curves following the procedure defined by Brainpool. Section 5 provides an example of the elliptic curve generation process using GCEC. Section 6 details the results of the tests and offers information about the implementation performance. Finally, Section 7 summarizes the most relevant conclusions and provides additional comments about the security of the analysed procedure.
2. Elliptic curves

An elliptic curve $E$ defined over a field $F$ is a plane non-singular cubic curve with at least one rational point [13]. Such a curve is defined by the following equation, known as the Weierstrass equation in non-homogeneous form [14]:

$$E : y^2 + a_1 xy + a_3 y = x^3 + a_2 x^2 + a_4 x + a_6,$$

where $a_1, a_2, a_3, a_4, a_6 \in F$ and $\Delta \neq 0$, being $\Delta$ the discriminant of the curve $E$ [15]. Condition $\Delta \neq 0$ assures that the curve is smooth, i.e., there are no curve points with two or more different tangent lines.

The Weierstrass equation can also be described using homogeneous coordinates, producing the following equation:

$$Y^2 Z + a_1 X Y Z + a_3 Y Z^2 = X^3 + a_2 X^2 Z + a_4 X Z^2 + a_6 Z^3.$$

The homogeneous version of the Weierstrass equation implies the existence of a special point, called the point at infinity, which is denoted as $O$ and does not have a counterpart in the affine plane.

Regarding the points of an elliptic curve, it is possible to define the following operations:

- Point addition: $Q + R = S$.
- Point doubling: $Q + Q = 2Q$.
- Scalar multiplication: $kQ = Q + \cdots + Q$ ($k$ times).

The set of points that satisfy the Weierstrass equation (plus the point at infinity), together with the point addition operation, fulfill the requirements to form a commutative group. This algebraic structure permits to use the elliptic curves in cryptography in a reliable way.

When working with finite fields of $q = p^m$ elements, where $p$ is a prime number and $m \geq 1$, it is possible to obtain simplified versions of the Weierstrass equation. If the finite field is a prime field (and it is important to point out that the Brainpool procedure only uses prime fields), i.e., $\mathbb{F}_q = \mathbb{F}_p$, where $p > 3$ is a prime number, the equation defining the elliptic curve becomes

$$y^2 = x^3 + ax + b,$$

where $a, b \in \mathbb{F}_p$, and $x$ and $y$ are the affine coordinates of a point that belongs to the elliptic curve.

One of the elements that determine the cryptographic strength of an elliptic curve is its number of points, also known as the cardinality or the order of the elliptic curve. This value is formed by all the points that satisfy the elliptic curve equation plus the point at infinity $O$.

If the field used to define the curve is a finite field, then the order of the curve is also a finite number. In a first approach to determine the cardinality of an elliptic curve, the Hasse theorem [16] states that the order of an elliptic curve defined over a prime field with $q$ elements must fall inside the interval $|q + 1 - 2\sqrt{q}, q + 1 + 2\sqrt{q}|$.

Several algorithms have been proposed during last years to compute the number of points of an elliptic curve. The most well known procedures are the Schoof [17] and SEA (Schoof-Elkies-Atkin) [18] algorithms.

The Schoof algorithm was proposed by René Schoof in 1985, and it is a polynomial-time algorithm which uses the Frobenius endomorphism and division polynomials. Its time complexity is $O(\log^6 q)$, though using fast polynomial and integer arithmetic it can be reduced to $O(\log^5 q)$. It is recommended for elliptic curves of relatively small bit lengths.

The SEA algorithm is an improvement of the Schoof algorithm, with enhancements devised by Elkies and Atkin. This algorithm has a time complexity of $O(\log^6 q)$, though it can be reduced to $O(\log^4 q)$ using fast arithmetic.

In order to avoid some possible attacks, the number of points of an elliptic curve should have a small cofactor (typically 1, 2, 3 or 4) [13]. The cofactor is the additional factor that accompanies the large prime factor, so their multiplication matches the order of the curve. If the cofactor is 1, then the order of the elliptic curve is a prime number.

Table 1 provides a comparison between RSA and ECC key lengths, with data taken from [15] and [19], where the security level must be interpreted as the cryptographic strength provided by a symmetric encryption algorithm using a key of $n$ bits. As it can be observed, the ratio between the key length in RSA and ECC clearly grows, which means that ECC is best adapted for applications where higher security levels are needed.

<table>
<thead>
<tr>
<th>Security level (bits)</th>
<th>RSA key length (bits)</th>
<th>ECC key length (bits)</th>
<th>Approximate ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>1024</td>
<td>160-223</td>
<td>4.6:6.4</td>
</tr>
<tr>
<td>112</td>
<td>2048</td>
<td>224-255</td>
<td>8:0.9:1</td>
</tr>
<tr>
<td>128</td>
<td>3072</td>
<td>256-283</td>
<td>10.8:12.0</td>
</tr>
<tr>
<td>192</td>
<td>7680</td>
<td>384-511</td>
<td>15.0:20.0</td>
</tr>
<tr>
<td>256</td>
<td>15360</td>
<td>512-571</td>
<td>26.9:30.0</td>
</tr>
</tbody>
</table>

3. Main characteristics of the Brainpool procedure

3.1 Key length

In ECC, the term key length is interpreted as the number of bits needed to represent the prime number $p$. The key lengths allowed by Brainpool are 160, 192, 224, 256, 320, 384, and 512 bits ([12], page 6).

3.2 Seed generation

The seeds used in Brainpool are generated in a systematic and comprehensive way. These seeds have been obtained as the first 7 substrings of 160 bits each of the number $π \cdot 2^{1120}$ = Seed_p_160 || . . . || Seed_p_512 | Remainder, where || denotes the concatenation operator ([12], page 24).
3.3 Seed to candidate conversion

Brainpool uses SHA-1 ([12], page 22) during the process of finding candidates for the parameters $p$, $a$, and $b$ given in formula (1). As the output of SHA-1 is 160 bits, and for different curves the length of the resulting parameters must be necessarily different, Brainpool performs a loop concatenating several SHA-1 outputs until the concatenated number has the proper bit length ([12], pages 22 and 24).

In addition to that, Brainpool uses two functions to generate the candidates, one for $p$ and another for $a$ and $b$. Those functions are very similar, in fact the only difference is that the most significant bit of $a$ and $b$ is forced to be 0 ([12], page 24). Given that another requirement states that the most significant bit of $p$ must be 1 ([12], page 23), this implies that the values $a$ and $b$ generated are such that $a, b < p$.

3.4 Validation of parameters $a$ and $b$

In Brainpool, once the algorithm has determined the value of $p$, it starts searching the proper values of the elliptic curve parameters $a$ and $b$. When a candidate pair is found, the resulting curve is tested against the security requirements. In case the curve is rejected, both $a$ and $b$ are discarded, starting a new search for a proper pair ([12], page 25).

3.5 Cofactors

As it was mentioned in Section 2, in order to generate cryptographically strong elliptic curves it is necessary to compute the number of points of the elliptic curve and to determine if that value is a prime number, or if it has a small cofactor. In this regard, the Brainpool specifications only allow curves whose number of points is a prime number ([12], page 6).

3.6 Factorizations

Two of the security requirements defined by Brainpool imply the factorization of integers. In one case, it is necessary to factorize the value $q - 1$, where $q$ is the order of the elliptic curve, in order to avoid attacks using the Weil or Tate pairings. Those attacks allow the embedding of the cyclic subgroup of the elliptic curve into the group of units of a degree-$l$ extension field of $\mathbb{F}_p$, where subexponential attacks on the Discrete Logarithm Problem (DLP) exist ([11], page 5). In the other case, the specification requests to factorize the value $d$, which is the square-free factor of $4p - u^2$, where $u = q - p - 1$, so it can be checked that the class number of the maximal order of the endomorphism ring of the elliptic curve is larger than $10^7$ ([11], page 5).

As the factorization of really big integers (such as those used in the aforementioned requirements) is a highly demanding computational task, it is recommended to use programs specifically designed for this mission that implement the latest advances in factorization.

4. GCEC application description

GCEC is a Java application that implements the Brainpool elliptic curve generation procedure as specified in [11] and [12]. GCEC consists of a single window panel, as it is shown in Figure 1.

In order to interact with the user, GCEC includes different elements such as combo boxes, buttons, and text boxes. Those elements are described next:

- Combo boxes:
  - **Length**: Length in bits of the prime number $p$ that defines the underlying finite field.
  - **Cardinality**: Algorithm used to compute the number of points of the elliptic curve. In this version of GCEC the user can choose between two algorithms: Schoof [17] and SEA (Schoof-Elkies-Atkin) [18], introduced in Section 2. The implementation of both algorithms, developed by Mike Scott, is part of MIRACL (Multiprecision Integer and Rational Arithmetic Cryptographic Library), a C library offered by CertiVox as free software, and licensed under the FOSS (Free and Open Source Software) approved AGPL (Affero General Public License) terms [20].

While the Schoof algorithm is coded into a single executable file (schoof.exe), the SEA algorithm requires three different executable files (mueller.exe, process.exe, and sea.exe) and a data file (mueller.raw) which contains the modular polynomials that are needed in the computations.

In addition to outperform the Schoof algorithm when working with elliptic curves of significant key lengths, the SEA implementation is able to...
compute the value associated to the order of the curve modulo the prime numbers included in a input file and, when enough of those results are gathered by the algorithm, it can determine the order of the curve. This is an important feature that is used by GCEC to speed up the calculations, as it is not necessary to complete the computation of the cardinality of the curve if it is detected that the order is divisible by any number other than 1.

- **Factoring**: External program used for the factorization of integer numbers. In this version of GCEC the user can choose between two open source factoring applications: yafu v1.33 [21] and Msieve v1.5 [22]. The source code of both applications is accessible through their SourceForge repositories. According to its author, yafu is “an interactive command line utility for integer factorization. It implements multi-threaded NFS, SIQS, and ECM as well as P+1, P-1, SQUFOF, Pollard’s Rho, and Fermat’s method. It also contains a very fast implementation of the Sieve of Eratosthenes” [21]. Besides, Msieve is “a C library implementing a suite of algorithms to factor large integers. It contains an implementation of the SIQS and GNFS algorithms”, in his author’s own words [22].

The reason for providing two different external factoring tools is that, when working with integers of more than 100 digits, the factorization time for any given integer may vary a lot between the two applications. Even for the same application, obtaining the factorization of two integers of the same bit length may differ substantially, as the internal routines of both applications are very sensitive to the nature of the specific number to be tested.

- **Text boxes**:
  - **Seed for P**: Initial seed that is used to compute the prime number \( p \) that defines \( F_p \).
  - **Seed for A & B**: Initial seed that is used to compute the curve parameters \( a \) and \( b \).
  - **Data**: Output produced by the program.

- **Buttons**:
  - **Process**: Computes all the curve parameters using the seeds for \( p, a, \) and \( b \).
  - **Stop**: Interrupts the computation without the possibility to continue the process.
  - **Save**: Stores the result of the computation in a file.
  - **Clear seeds**: Deletes the text boxes that contain the seeds.

Selecting a specific curve length automatically fills in the text boxes associated to the seeds for the parameters \( p, a, \) and \( b \), so if the user wants to try the seeds provided by Brainpool it is not necessary to alter those values. On the other hand, if the user wants to try a different pair of seeds, he must enter the new values in the text boxes after clicking the **Clear seeds** button.

5. Generating an elliptic curve with GCEC

This section provides a complete example on how to generate an elliptic curve using GCEC.

After launching the application, the first step consists in setting the combo box options. In the example provided (see Figure 1), GCEC generates the Brainpool curve of 160 bits using the SEA algorithm and the yafu factoring application.

Once the user has clicked the **Process** button, GCEC starts with the computations, displaying information in the **Data** text box whenever new data is generated. A summary of the options selected by the user in the four combo boxes is shown in the **Data** text box (see Figure 1).

Clicking the **Process** button enables the **Stop** button (and, at the same time, the **Process** button gets disabled). If the user clicks the **Stop** button, the procedure is interrupted and the user is informed of that event in the **Data** text box (see Figure 2).

![Fig. 2: Elliptic curve generation process stopped by the user.](image)

Whenever GCEC finishes computing the elliptic curve, the **Stop** button is disabled and both the **Process** and **Save** buttons are enabled (see Figure 3).

Although the information displayed in the **Data** text box can be copied and pasted into any text editor, the user can store the result of the computation in an easy way by clicking the **Save** button, which pops up a dialog so the user can select the file where the information will be stored.

6. Tests and results

The tests whose results are presented in this section were completed using a PC with Windows 7 Professional OS
and an Intel Core i7 processor at 3.40 GHz. As the i7 processor has 4 core processors and 2 logical processor per core processor, the end result is that the operating system manages 8 processors. For applications that are prepared to run along multiple threads, such as yafu and, to least extent, Msieve, this implies an important reduction of the working time.

Table 2 includes the number of candidates for the parameters $p$, $a$, and $b$, together with the number of elliptic curve point counting operations. The number of integer factorizations performed in all the tests is 2.

Table 3 includes the running time obtained when executing the GCEC application in the testing computer with the Brainpool sample curves. The term N/A stands for not available, in the sense that the test referred to was launched two months before this contribution was prepared and, at the time of writing these lines, was still in progress.

Figure 4 shows graphically the running time in seconds of the Brainpool tests with key lengths up to 384 bits. Besides, Figure 5 presents the running time for all the Brainpool key lengths.

7. Conclusions

This report has analysed the elliptic curve generation procedure defined by Brainpool. In order to generate new elliptic curves that fulfill the security requirements of that organization, a Java application has been developed by us following the indications of the Brainpool specifications. In addition to that, the application can also be used for testing the sample elliptic curves included in those recommendations.

As it was expected, the execution time increases for bigger
key lengths (320, 384, and especially 512 bits). As it was mentioned in Section 2, a key length of 256 bits in ECC is equivalent to a key of 112-128 bits in a symmetric algorithm and of 2048-3072 bits in RSA, whilst a key length of 512 bits in ECC offers the same strength as a symmetric key of 192-256 bits and an RSA key of 7680-15360 bits.

For key lengths up to 256 bits, most of the computation time is dedicated by the program to execute the SEA algorithm. In comparison, for key lengths bigger than 256 bits, most of the time is devoted to factorize the integers of each test.

Regarding the comparison of Msieve and yafu, from the results of the tests it is clear that even though for small key lengths both programs offer a similar performance, when working with bigger key lengths (starting in 320 bits), yafu provides a significant improvement over Msieve when both applications are executed multi-threaded.

Acknowledgment

This work has been partially supported by Ministerio de Ciencia e Innovación (Spain) under the grant TIN2011-22668.

References