IMPLEMENTING ELLIPTIC CURVE CRYPTOGRAPHY

Design of a Standard Compliant Java Library

Wolfgang Bauer
Institute for Applied Information Processing and Communications
Graz University of Technology
wolfgang.bauer@iaik.at

Abstract Properties like short keys and efficient algorithms make elliptic curve cryptography (ECC) more and more interesting for future oriented applications. In this paper we give a short overview of the basics of ECC. Thereby we show where programmers can gain possible speed-ups and what parts are crucial. Since there are many different implementation options and some of the algorithms are patented, we believe that there is no optimal solution. Therefore we introduce a software framework, which allows a transparent replacement of data and algorithms. Furthermore, we discuss aspects of the standardized encoding, and point out where interoperability problems could occur.

Keywords: ECC, Java

INTRODUCTION

The security of public key cryptography is based on hard mathematical problems. Today, the popular algorithms are either based on the integer factorization problem (IFS), like RSA, the discrete logarithm problem (DLP), like DSA, or the elliptic-curve discrete-logarithm problem (ECDLP). The only problem where no sub-exponential time algorithm is known so far is the ECDLP. That is why the key lengths of elliptic-curve based algorithms are much shorter and increase much slower over time. The Austrian signature ordinance, for instance, stipulates RSA keys of at least 1023 bits, whereas ECDSA keys may be 160 bits only (by the end of the year 2005).

Therefore ECDLP based algorithms getting more and more important especially for smart cards, which are very limited in memory. For this

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35612-9_23

B. Jerman-Blaži et al. (eds.), *Advanced Communications and Multimedia Security*

reason the Austrian Social Insurance Smart Card, which every Austrian citizen should have at the beginning of the year 2003, uses the Elliptic Curve DSA (ECDSA) for secure electronic signatures.

In this paper we introduce design aspects and problems of a standard compliant ECC Java library. This library is designed to work not only for signature creation, but also for verification. This means, that it must be able to deal with all requirements defined in the standards. There are numerous of papers which discuss different aspects and algorithms of ECC, whereas this paper focuses on implementing all of the standard requirements. We point out software design decisions and where interoperability problems may occur.

The rest of this paper is organized as follows. The next section provides a short overview of elliptic-curve cryptography. Afterwards we discuss components and implementation options of an ECC software system. Section 3 defines the design goals and section 4 introduces the actual design of an ECC Java library. Finally conclusions are given.

1. BASICS OF ELLIPTIC CURVE CRYPTOGRAPHY

The idea of ECC was first introduced by Neil Koblitz and Victor Miller in the mid eighties. Since then cryptographers started to analyze the ECDLP and so far no vulnerabilities have been discovered. First ECC was an academic concern but in recent years it started to gain commercial interest, which lead to standards like [7] and [8].

To understand ECC you have to know a lot of math, which is usually scaring for programmers. Therefore one common way is to start with the following graphical example. Figure 1 shows an elliptic curve ($y^2 = x^3 + ax + b$) defined over the field of real numbers ($a, b, x, y \in \mathbb{R}$).

We define a basic operation, the point addition. The sum of two points is determined as shown in figure 1. Draw a line through the points P and Q and the intersection of this line with the curve is the Point -(P+Q). Now take the negative y-coordinate (which again is a point on the curve) and you are done. For doubling a point use the tangent and proceed as with the point addition. This rule obviously will not work if the line through the point(s) is parallel to the y-axis. Therefore we define the point at infinity ($\mathcal{O}$) as a result of this operation.

Calculations with infinite fields, as with $\mathbb{R}$ in the example above, are not suitable for cryptographic purposes. That is why finite fields F_q, as described in section 2.1, are used. The set $E(F_q)$ consists of all points $(x, y), x \in F_q, y \in F_q$ satisfying the elliptic curve equation and the point $\mathcal{O}$.

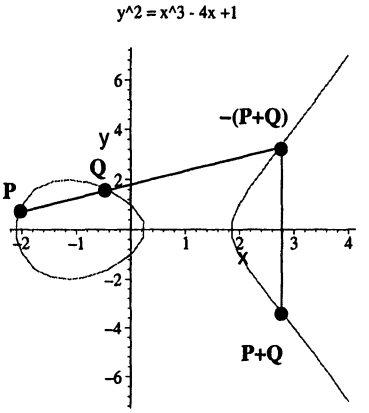

Figure 1. Elliptic Curve Point Addition

Table 1. Point Addition and Doubling Formulas
$P = (x_1, y_1)$ *and* $Q = (x_2, y_2) \in E(F_q)$; $P \neq Q$

$P + Q = (x_3, y_3)$	$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2$
	$y_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3) - y_1$
$2P = (x_3, y_3)$	$x_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1$
	$y_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)(x_1 - x_3) - y_1$

Table 1 shows the formulas of the previously described point operations. It can be shown that $E(F_q)$ together with the point addition forms a group. We write the "+" for the group operation (point addition) and the neutral element is $\mathcal{O}$. Thus $\mathcal{O} + P = P + \mathcal{O} = P \; \forall \; P \; of \; E(F_q)$.

With this additive group [9] defines the ECDLP problem as follows. Given an elliptic curve E defined over a finite field F_q, a Point $P \in E(F_q)$ of order n, and a point $Q = lP$ where $0 \leq l \leq n-1$, determine l.

Based on this hard mathematical problem one can adapt algorithms based on the DLP to ECDLP by replacing $\mathbb{Z}_p^*$ with the elliptic curve group. After this short introduction to the EC arithmetic, the next section shows the basic components of a software implementation.

2. COMPONENTS AND IMPLEMENTATION OPTIONS

There are many other papers, like [5] [6] [1], comparing and explaining aspects of elliptic curve crypto systems. Therefore this section only points out where software developers have various implementation options and what has to be heeded. According to our software design, introduced in section 4, this section is subdivided into finite field arithmetic, EC arithmetic, and data formatting options.

2.1 Finite Field Arithmetic

Finite fields form the basis of EC arithmetic. As the formulas of table 1 show, one simple point addition requires many operations in the underlying finite field. Therefore special attention and efficient algorithms are required.

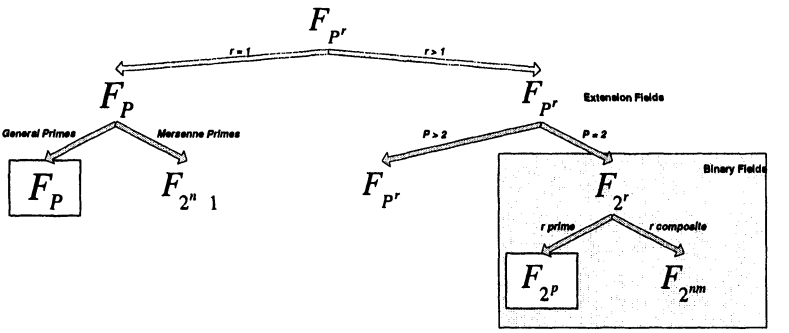

Figure 2. Classification of Finite Fields

Figure 2 shows a classification of finite fields. The two boxed leafs of this tree (F_P and F_{2^p}) are used for ECC and are specified in the standards.

2.1.1 Prime Fields. Prime fields are the most efficient choice for software implementations. Many papers like[6] describe algorithms for long integer arithmetic. The basic addition and multiplication operations use conventional integer arithmetic based on the processor's word size. Therefore they are very fast.

The crucial operations are the reduction modulo a prime and the calculation of the multiplicative inverse element. By the use of projective coordinates the inversion in the formulas of table 1 can be avoided. Only one inversion at the end of the ECC calculation is needed to regain the affine coordinates. Therefore the costs for this operation can nearly be neglected. The major design decision in prime field arithmetic is how to integrate the modular reduction and how to represent field elements (e.g. use a Montgomery or Barret reduction).

As a last point we should mention, that in general the square operation can be done more efficient than a multiplication. This is the reason why we decided to put this operation into the list of basic field operations, as show in section 4.

2.1.2 Binary Fields. There are several ways of representing field elements in F_{2^p}, whereby polynomial basis representation seems to be the most efficient [6]. Unfortunately, to be standard compliant, one must be able to deal with Gaussian Normal Basis (GNB) representations as well. Since almost every algorithm in GNB seems to be patented and this representation does not provide any benefits, we decided not to support it at all. Nevertheless, we still have the option to implement a base transformation and therewith gain standard compliance subsequently.

In polynomial base representation the field elements are considered as polynomials with coefficients $\in F_2$. Addition, which is the same as subtraction, is a simple XOR operation. Multiplication is a conventional polynomial multiplication modulo an irreducible polynomial.

Microprocessors do not support polynomial arithmetic and therefore software implementations of F_{2^p} multiplication are usually slow. However, modulo reduction can be performed rather efficient. That is especially true for trinomials and pentamonials, which are the standardized reduction polynomials. In binary fields the programmer has to consider what algorithms to use and data formats that allow efficient polynomial arithmetic. As in prime fields squaring can be done more efficient.

2.2 EC Arithmetic

Efficient finite field arithmetic is the key for a high performance ECC implementation. Even so the EC group arithmetic offers a lot of tuning possibilities. The basic operation for ECC is the computation of the scalar product $k * P$ $0 \leq k \leq n-1$, which is a repeatedly application of the basic group operation. Depending on the field type (prime or binary) the formulas are a little bit different. The actual group for the calculation is determined by a set of so called domain parameters. It consists of the following items.

- The field size, which defines the underlying field F_q. In case of binary fields the used basis type and in case of a polynomial base representation the irreducible polynomial, is required.
- The Curve Parameters a and b $\in F_q$
- The Base Point G = (x_G, y_G), $x_G, y_G \in F_q$ on E of prime order.
- The Order n (of the base point G).
- The cofactor h = $\#E(F_q)/n$
- Optional parameters.

Depending on the actual domain parameters EC arithmetic offers some implementation variants. For instance there exist some elliptic curves, where more efficient algorithms are known (e.g. Koblitz curves). Further speed-ups can be achieved by the fact that in elliptic curve groups one gets the negative element nearly for free. This allows the construction of efficient addition-subtraction chains for the scalar multiplication [4]. The point representation influences the overall performance as well, and the type of coordinates (affine, projective or mixed) to be used, has to be considered.

2.3 Algorithms

All the previously mentioned parts form the basis for an elliptic curve crypto system. Algorithms originally based on the discrete logarithm problem can be adapted to work on elliptic curves. Though their exist other ECC algorithms like an EC version of the Diffie-Hellman key-agreement scheme or encryption schemes, the most famous EC algorithm is the ECDSA. It can be subdivided in 3 steps.

1 Message Digesting: using SHA-1

2 EC Computation: calculation of k*G (k randomly chosen)

3 Modular Computations: modulo n

The third point always requires some arithmetic modulo a prime. Therefore it is not possible to calculate an ECDSA signature with binary field arithmetic only. But this fact is mainly important for signature creation devices.

2.4 Data Formatting and Standards

The sections above have shown how elliptic curve crypto systems operate. To use them in praxis and to ensure interoperability one has to present the data in a system independent and standartized way. Several ECC standards exist and figure 3 shows their compatibility.

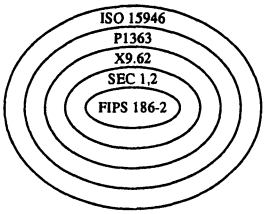

Figure 3. Compatibility of ECC Standards

To make sure an ECDSA signature can be verified, the other party must know the public key and the related domain parameters. Therefore the standards define structures for the encoding of all the data. There are two drawbacks of this approach.

- Memory requirements: the size of the encoded domain parameters is much larger than the signature and the keys.
- The verifying party must either rely on the validity of the domain parameters or perform some checks.

To circumvent this unsatisfying situation the possibility exists to use a unique object identifier. This approach will only work if the domain parameters are standardized. [7] [8] and [3] altogether define OIDs for 20 curves over prime fields (112 - 521 bits) and for 38 curves over binary fields (113 - 571 bits). We believe that this variant is the best, since there is no argument to use other curves than these standardized. Application do not have to do a parameter validation and the encoding size is negligible. One disadvantage is, that every party must have a table, assigning OIDs to domain parameters.

There is a third variant, where domain parameters are omitted at all. The application either implicitly knows them, or within a public

key infrastructure, these domain parameters may be inherited from a Certification Authority. This approach might be useful in some special cases, but in general it could cause interoperability troubles.

Patented algorithms might cause further interoperability problems. Unfortunately some of them are contained in the standards. For instance the following three ways of encoding elliptic curve points are standardized.

- The uncompressed form, which consists of the x and y coordinate of this point.
- The compressed representation only contains the x coordinate and one single bit indicating which of the two possible points to take. To decompress a point, one has to solve the elliptic curve equation with a given x value, which is a quadratic equation.
- A hybrid form, which is a mixture of the points above.

Certicom holds a patent on the point compression and thus not every software might support this feature. Nevertheless a good software design allows all of the encoding possibilities above. To take patent issues into consideration the design should support plug-able modules and easy configuration.

A further point that should be considered is the encoding of the data structures, like domain parameters and public keys. Until today the most common format is the Abstract Syntax Notation 1 ASN.1. The ECC standards define such ASN.1 structures und thereby allow ECDSA to be used with existing X.509 certificates (private key encoding for use in PKCS#8 is only defined in [2]). As an alternative the Extensible Markup Language XML is getting more and more important nowadays. But the syntax for ECDSA with XML signatures is only a draft yet. Nevertheless software designers should keep in mind that other formatting options may be desired in the future.

3. DESIGN GOALS

As already stated in the introduction, the software design is very much dependent on the actual deployment. For instance, software for signature creation devices usually has to implement only a small part of the corresponding standard. That means one can optimize code for one set of domain parameters (see section 2) and choose one of the encoding possibilities. Furthermore, the signature creation devices may leave some tasks to the environment. Memory and computational limited devices may just compute the signature value and the formatting and encoding may be left to the application.

In this paper we focus on the more general case, which deals with both, signature creation and verification. The challenge is to write code, which provides for integration of all standardized features. Since there are many implementation options and design tradeoffs there is not one best way. Furthermore, new faster algorithms might pop up and subsequent integration should not be problem. Finally, patent issues have to be considered and thus algorithms and data representations should be plug-able. The main points of our design guidelines can be summarized with the following key words.

- Modularity: Subdivide the problem and make the modules independent.
- Extensibility: Easy integration of new algorithms and data representations.
- Maintainability: The code should be well structured and easy to understand.
- Performance: At first view a large software framework (with abstract classes and interfaces) and high performance seem to be contradictory design goals. However performance is primary influenced by the used algorithms and therefore a flexible and clear design pays off.
- Robustness: Ensure a well defined behavior particularly in case of an error.

After this definition of the design goals, the next section introduces the software framework. We will pick out some parts and explain how some of the points above can be achieved.

4. SOFTWAREDESIGN

The design of the whole library would go beyond the scope of this paper. Therefore this section introduces the overall module concept and explains some features in more detail.

The main parts of the library are shown in Figure 4. According to the design criteria above, each module operates independent. The behavior of the modules is defined by interfaces and the actual implementation is not visible from outside. This design allows a transparent replacement of the algorithms and data representations.

Finite Field Arithmetic. The common interface for all finite fields defines the basic field operations like:

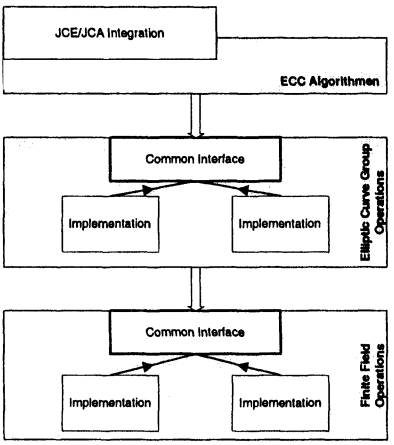

Figure 4. Software Module Design

```
public interface Field {
  public FieldElement getONEelement();
  public FieldElement getZEROelement();
  public void negate(FieldElement a);
  public void add(FieldElement a, FieldElement b);
  public void invert(FieldElement a);
  public void multiply(FieldElement a, FieldElement b);
  public void square(FieldElement a);
  ...
}
```

As you can see, there are no implementation specific elements within this interface. The square operation, which is not really required, was added for performance reasons. The interface contains some additionally methods to get the field type and some other utility functions that are not listed above. Furthermore, the finite field module contains interfaces for field elements and their values. Thereby implementations can choose any desired data structures and algorithms.

Elliptic Curve Arithmetic. The structure of this module is very similar to the previously presented. Again the developer has the complete freedom of how to present the data and which algorithms to use. Depending on patent issues it is possible to change algorithms without any modification in the rest of the whole library. Also the coordinate

types (affine, projective or mixed) are not hard coded and can be changed transparently. The common interface offers methods for the basic group operation. To speed up the ECDSA signature verification process we decided to put a method for simultaneous multiplications ($P = h_1G+h_2W$) into the interface (see [10]).

Elliptic Curve Algorithms. This module contains the ECC algorithms like ECDSA or ECDH. It defines interfaces for private and public keys. Furthermore the administration of domain parameters and encoding specific tasks are placed here. Applications access the library through the Application Programming Interface API. The next session discusses the integration of this API with the Java framework.

Java Integration. Security has been one of the main design goals of the Java programming language. Therefore it offers a large framework for cryptographic algorithms called Java Crypto Extension (JCE)/Java Crypto Architecture (JCA). The so-called provider concept allows third party implementation of various algorithms but until now no framework for ECC exists. One can use the existing signature interface to work with ECDSA but there is no common way to specify the domain parameters and keys. Therefore, always some proprietary code must be included in the application.

A further shortcoming of the Java framework is the design of the BigInteger class. This class is usually used for arbitrary long integer arithmetic. Unfortunately it has two serious disadvantages for the use in ECC.

- Instances of this class (objects) are immutable. This means, for every finite field operation a new object must be created. Since ECC computations require many operations in the underlying finite field, many of the BigInteger objects have to be created, which is not very efficient.

- There is no public method for the often used square operation and therefore the slower multiply method has to be taken.

For the reasons above, it makes sense to implement an own, mutable BigInteger class.

5. CONCLUSIONS

Short keys and efficient algorithms make ECC interesting for future oriented applications. This paper provides an introduction into ECC. We have discussed various parts of a standard compliant software library,

like finite field and EC arithmetic. The numerous implementation variants and encoding options, as well as possible patent issues, require a flexible software architecture. We have introduced a design, which enables programmers to transparently add, remove, or replace algorithms and data types. This framework might be considered as a performance penalty. Usually the main speed-ups can be achieved by efficient algorithms and data structures and therefore the advantages of a plug-able design pay-off. Furthermore, a clear and modular structure provides for easy integration of new algorithms guarantees the maintainability.

References

[1] Michael Brown, Darrel Hankerson, Julio Lopez, and Alfred Menezes. Software implementation of the NIST elliptic curves over prime fields. In *CT-RSA*, pages 250–265, 2001.

[2] Certicom. Sec 1: Elliptic curve cryptography, 2000.

[3] Certicom. Sec 2: Recommended elliptic curve domain parameters, 2000.

[4] Daniel M. Gordon. A survey of fast exponentiation methods. *J. Algorithms*, 27(1):129–146, 1998.

[5] Jorge Guajardo and Christof Paar. Efficient algorithms for elliptic curve cryptosystems. In *CRYPTO*, pages 342–356, 1997.

[6] Darrel Hankerson, Julio Lopez Hernandez, and Alfred Menezes. Software implementation of elliptic curve cryptography over binary fields. In *Cryptographic Hardware and Embedded Systems*, pages 1–24, 2000.

[7] IEEE. 1363 standard specification for public key cryptography, 2000.

[8] American National Standards Institute. X9.62-1998, public key cryptography for the financial services industries: The elliptic curve digital signature algorithm (ecdsa), 1998.

[9] D. Johnson and A. Menezes. The elliptic curve digital signature algorithm (ecdsa), 1999.

[10] Bodo Moeller. *Algorithms for Multi-exponentiation*, pages 165–180. Springer-Verlag, 2001.