ELLIPTIC CURVE CRYPTOGRAPHY. JAVA PLATFORM IMPLEMENTATIONS

V. Gayoso Martínez[1], L. Hernández Encinas[1], C. Sánchez Ávila[2]

[1]Departamento Tratamiento de la Información y Codificación
Instituto de Física Aplicada. CSIC, Madrid
{victor.gayoso, luis}@iec.csic.es
[2]Departamento de Matemática Aplicadas a las Tecnologías de la Información
ETSI Telecomunicación, Univ. Politécnica de Madrid, Madrid
csa@mat.upm.es
Spain

Abstract: Elliptic Curve Cryptography (ECC) is one of the most interesting systems for protecting sensitive information nowadays. The latest versions of the Java Platform include classes and interfaces making ECC available to programmers, but due to the nature of Java it is still necessary to employ additional third party packages in order to use cryptographic operations and procedures related to ECC. In the present work, an extensive review of the most important ECC implementations in Java is presented.

Key words: Elliptic Curve Cryptography, Java Platform, ECDSA, ECDH, ECIES.

1. INTRODUCTION

1.1. Public key cryptography

Since the development of public key cryptography by Whitfield Diffie and Martin Hellman in 1976 [1], several cryptosystems have been proposed. The most important features to be requested to a cryptosystem are security and efficiency, and in general, both characteristics depend on the mathematical problem on which it is based. The list of problems that are considered computationally infeasible to solve includes the integer factorization problem (IFP), the discrete logarithm problem (DLP), and the elliptic curve discrete logarithm problem (ECDLP).

In 1985, Victor Miller [2] and Neal Koblitz [3] independently proposed a cryptosystem based on elliptic curves, whose security relies on the ECDLP problem. Elliptic Curve Cryptography (ECC) can be applied to data encryption and decryption, as well as to the creation and verification of digital signatures.

This paper has been presented in the Plenary Session of the International Conference on Information Technologies (InfoTech-2009), Bulgaria, 17-20 September 2009.

As in the case of IFP and DLP, no algorithm is known that solves the ECDLP in an efficient way. Moreover, the ECDLP is regarded by some authors as the hardest of these three problems. Although some operations take more time in the ECC system compared to other public key systems, as the key size is smaller in ECC, some studies suggest that there are no practical differences in performance between ECC and RSA. A comparison among the RSA/DSA/ECC cryptosystems and their key lengths is shown in Table 1.

Table 1. Key length comparison of RSA/DSA and ECC cryptosystems

MIPS year	*RSA/DSA key length*	*ECC key length*	*Ratio*
10^4	512	106	5:1
10^8	768	132	6:1
10^{11}	1024	160	7:1
10^{20}	2048	210	10:1

1.2. Elliptic curve cryptography

An elliptic curve E over the finite field (or Galois Field) GF is defined by the following equation known as a Weierstrass equation [4]:

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{1}$$

where a_1, a_2, a_3, a_4, $a_6 \in GF$ and $\Delta \neq 0$, being Δ the discriminant of E that can be calculated in the following way [5]:

$$\left.\begin{aligned}
&\Delta = -d_2^2d_8 - 8d_4^3 - 27d_6^2 + 9d_2d_4d_6\\
&d_2 = a_1^2 + 4a_2\\
&d_4 = 2a_4 + a_1a_3\\
&d_6 = a_3^2 + 4a_6\\
&d_8 = a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2
\end{aligned}\right\} \tag{2}$$

Condition $\Delta \neq 0$ assures the curve is "smooth", i.e., there are no curve points with two or more different tangent lines.

However, the Weierstrass equation is not used in practice. Instead of it, depending on the characteristic of GF and the value of a_1, it is possible to work with the following simplified equations:

$$\left.\begin{array}{lll}
\text{Characteristic of } GF \neq 2{,}3, & & y^2 = x^3 + ax + b\\
\text{Characteristic of } GF = 2, & a_1 \neq 0, & y^2 + xy = x^3 + ax^2 + b\\
\text{Characteristic of } GF = 2, & a_1 = 0, & y^2 + cy = x^3 + ax^2 + b\\
\text{Characteristic of } GF = 3, & a_1^2 \neq -a_2, & y^2 = x^3 + ax^2 + b\\
\text{Characteristic of } GF = 3, & a_1 = -a_2, & y^2 = x^3 + ax + b
\end{array}\right\} \tag{3}$$

Two types of finite fields $GF(q)$ (with $q = p^m$ elements) are used in ECC: prime finite fields $GF(p)$ (where p is an odd prime and $m = 1$) and binary finite fields $GF(2^m)$ (where $p = 2$ and m can be any integer higher than 1). A comparison among the bit length of some cryptosystems and the size of the related finite field for elliptic curves is shown in Table 2, where |*p*| represents the bit length of the integer *p*.

Table 2. Key length according to FIPS 186-2

Key Length	***Example Algorithm***	**|*p*|**	***M***
80	*SKIPJACK*	*192*	*163*
112	*Triple-DES*	*224*	*233*
128	*AES-Small*	*256*	*283*
192	*AES-Medium*	*384*	*409*
256	*AES-Large*	*521*	*571*

1.3. Related standards

Either in the environment of RSA or ECC, any theoretical finding cannot be used directly, as it is necessary to define data structures and procedures to manage the information. Currently there are three immediate applications for ECC in cryptography, as it is described in next paragraphs.

Elliptic Curve Digital Signature Algorithm

FIPS 186-2 [6] describes all the algorithms and digital signature schemes that can be used by any agency of the U.S. government. Currently those algorithms are DSA, RSA and ECDSA. ECDSA is the elliptic curve variant of the Digital Signature Algorithm (DSA).

NIST and ANSI X9.62 state a minimum key size of 1024 bits for RSA and DSA and 160 bits for ECC, which provide an equivalent security to a symmetric block cipher with a key size of 80 bits. NIST has published a list of recommended elliptic curves for protection with different symmetric key sizes (80, 112, 128, 192, and 256 bits). In general, ECC over a binary field requires a key size twice that of the corresponding symmetric key. ECDSA scheme is also included in IEEE 1363 and SEC 1 standards.

As a comparison, texts signed with a 1024 bits RSA key produce a digital signature of 128 bytes, whilst the same text signed with a 192 bits ECDSA key generates a digital signature of 48 bytes.

Elliptic Curve Integrated Encryption Scheme

The most extended encryption/decryption scheme based on ECC is ECIES [7], being a variant of the ElGamal scheme. ECIES can be found at ANSI X9.63, IEEE 1363a and SEC 1 [8] standards.

ISO/IEC 18033-2 [9] includes an enhanced version identified as ECIES-KEM, which includes modifications in order to prevent “benign malleability” issues.

As an example, a symmetric key encrypted with a 1024 bits RSA key produces an output of 128 bytes compared with the output of 84 bytes if the encryption is performed with ECIES.

Elliptic Curve Diffie-Hellman

The main objective of key exchange protocols is to put in contact two or more entities communicating through an open and insecure channel, sharing a secret key that will provide data confidentiality and integrity to any information exchanged.

ECDH denotes the generic key exchange scheme based on the Diffie-Hellman mechanism applied to elliptic curves. Some practical implementations can be found at ANSI X9.63, IEEE 1363, NIST SP 800-56A and SEC 1 documents.

1.4. Commercial adoption

On February 16th 2005, the NSA (National Security Agency) announced that it had decided to adopt ECC as part of a U.S. government standard in order to improve the secure management of sensitive-but-unclassified information. The NSA has recommended a group of algorithms called Suite B, including ECMQV, ECDH, and ECDSA.

2. PROBLEM DEFINITION

Since its appearance in the mid-1990s, the Java language has experienced a constant growth regarding the number of programmers and commercial deployments, being massively used in web and corporate applications. As a result of its continuous evolution, several versions targeting specific platforms have appeared: *Java Platform Standard Edition* (Java SE) for desktop computers, *Java Platform Enterprise Edition* (Java EE) for advanced servers, *Java Platform Micro Edition* (Java ME) for handsets and PDAs and Java Card (JC) for smart cards.

Regarding Java SE, as the naming syntax has changed during the last years, its version history is included hereafter in order to avoid mistakes when referencing the proper version:

- JDK 1.0 (1996)
- JDK 1.1 (1997)
- J2SE 1.2 (1998)
- J2SE 1.3 (2000)
- J2SE 1.4 (2002)
- J2SE 5.0 (2004)
- Java SE 6 (2006)

In the Java architecture, the Security API (built around the *java.security* package) is one of the most important interfaces of the language. The first version of the Security API for the JDK (Java Development Kit) 1.1 introduced the Java Cryptography Architecture (JCA), which allows the management of digital signatures and message digests.

In the following versions, Java SE extended the JCA functionality, including a provider architecture which allows multiple and interoperable cryptographic implementations. More specifically, Java Cryptography Extension (JCE) provides implementations for MAC algorithms and for encryption, key agreement and key generation methods. JCE was originally an optional package, but since J2SE 1.4 it is included in the core Java SE distribution. Algorithm independence is achieved by means of specific “engines” or cryptographic services that implement the security functionality.

Before J2SE 5.0, the JCA/JCE environment did not include specific classes for ECC. Users willing to use those algorithms were forced to use software from third parties that was not compatible with software from other vendors. From J2SE 5.0 release onwards, some classes and interfaces have been included in order to facilitate a standard ECC support. Those additions can be found at the *java.security* package. However, it is still necessary to use third party engines in order to access all the power that ECC can provide to Java applications. Therefore, the main problem now consists in selecting the proper third party module.

3. PROBLEM SOLUTION

There are several libraries and cryptographic modules in the market that can be used for the development of cryptosystems. Due to native code performance, most of the implementations are developed in C/C++ or directly in assembly language. However, as far as the aim of this paper is to review the state of the art of current (i.e. not abandoned projects) ECC implementations using the Java language, we will focus in two libraries: Bouncy Castle and IAIK.

3.1. Bouncy Castle

Legion of the Bouncy Castle, a group of volunteers and cryptography enthusiasts, has developed several Java implementations. Its latest release (1.43) includes a lightweight cryptographic API for Java and C#, a provider for the JCE and JCA, a clean room implementation of the JCE 1.2.1, a library for reading and writing encoded ASN.1 objects, and different generators/processors for X.509 certificates, S/MIME and CMS, OpenPGP, etc. The lightweight API is a vendor-specific set of APIs that implement all the underlying cryptographic algorithms, and it is intended to be used in memory constrained devices or when easy access to the JCE libraries is not possible.

Regarding ECC, Bouncy Castle supports both ECDSA and ECDH, creation of ECDSA certificate requests, encoding of public and private keys in accordance with SEC 1 and a draft implementation of ECIES. As a drawback, this version still does not include the ECIES-KEM variant. Two versions of key agreement protocols using ECC are supported, standard Diffie-Hellman key agreement and standard key agreement with co-factors.

The BouncyCastle package *org.bouncycastle.math.ec* consists of the following four classes:

1. The class *ECConstants*, which provides the numbers 0 to 4 as *BigIntegers*.
2. The abstract class *ECCurve,* which represents an elliptic curve in the Weierstrass normal form.
3. The abstract class *ECFieldElements,* which represents an element in the Galois field that is used.
4. The abstract class *ECPoint*, which represents the points on the elliptic curve and implements the arithmetic of this curve.

All the abstract classes are implemented by two subclasses derived from them, *Fp* and *F2m,* representing curves defined over $GF(p)$ ($p > 2$) and over fields of characteristic 2, respectively.

At the beginning of 2008 a study was made public [10] exposing a vulnerability in the implementation of the *ECPoint* class, which could be used in real attacks, for example against the ECIES scheme. It must be noted that this vulnerability was solved in Bouncy Castle v 1.33, being the open review process one of the benefits of open source.

The reader can find an example of signature creation and verification using the Bouncy Castle ECC library at http://www.bouncycastle.org.

Evaluating the level of support both in terms of encryption and signature operations and of supported programming platforms, and given its continuous evolution, Bouncy Castle is one of the best cryptographic packages developed as open source, not only for ECC but also for other algorithms as RSA, AES, etc.

3.2. IAIK

The *Institute for Applied Information Processing and Communications* of the Graz University has developed a set of cryptographic tools based on Java. Among those tools are IAIK-JCE (main module including RSA, DES, AES, Blowfish and other algorithms), IAIK-iSaSiLk (TLS 1.0 and SSL 3.0 implementations) and IAIK-ECC, to name only a few.

Release 2.18 of the *IAIK Elliptic Curve Cryptography (ECC) Library*, available since December 2008, presents the following features:

- It is compliant with ANSI X9.62 and IEEE 1363 standards.
- It allows creating and verifying ECDSA signatures and supports the ECDH key agreement scheme.
- ECDSA with SHA-2 support according to ANSI X9.62:2005 and BSI TR-03111 v1.0.
- Arithmetic over prime and binary finite fields. The prime field arithmetic is based on the *BigInteger* class.
- The binary field arithmetic uses polynomial base representation and includes a very generic implementation of the field operations.
- Elliptic curve arithmetic with affine and projective coordinates.
- ECDSA integration into the JCE/JCA architecture.
- ECDH/ECDSA integration into the JCE/JCA architecture with and without cofactor multiplication.
- ASN.1 encoding of signatures, public and private keys for usage with X.509 and PKCS#8.

Examples of a simple ECDSA signature and the ECDH protocol using the IAIK-ECC library can be found at http://jce.iaik.tugraz.at.

The IAIK-ECC library allows several customization options:

- Pre-computation of points: It is possible to first pre-compute and store a number of points to improve further key pair and signature generations. Pre-computation has no influence on signature verification and should be used if many key pairs or signature generations are performed on a given curve. By default no pre-computation is used.
- Encoding: The standards offer two variants on how to encode domain parameters. It is possible either to specify the object identifier OID (if it exists) or

encode the parameters explicitly. IAIK's ECC library supports both methods, but the default setting encodes the parameters explicitly.

• Point verification: If a certificate or any other ASN.1 encoded public key is parsed, the default implementation can optionally check if the decoded point is on the elliptic curve.

Evaluation and beta versions of the IAIK software can be downloaded, which makes it another good option for the development of cryptographic Java applications.

4. CONCLUSION

The main conclusions derived from the previous report can be summarized as follows:

- During the last years, Java has been one of the technologies with a fastest growth. Cryptographic capabilities were added first to the Java SE platform and then extended to other platforms such as Java Card.
- In Java SE, it is possible to use ECC implementations adapted to the JCA/JCE framework from version 5.0. However, it is also possible to use lightweight APIs provided by different suppliers, which can be a good option if the developer must work with Java SE versions prior to the 5.0 release.
- Among the independent implementations developed outside the Java standardization bodies, *Bouncy Castle* and *IAIK* outstand above the rest.
- Both of them provide high quality implementations and can be used not only in ECC applications but also in other types of cryptographic deployments.

Acknowledgement. This work has been partially supported by Ministerio de Industria, Turismo y Comercio (Spain), in collaboration with CDTI and Telefónica I+D under the project SEGUR@ (CENIT-2007 2004).

REFERENCES

[1] Diffie, W., Hellman, M.E. (1976). New directions in cryptography. *IEEE Transactions in Information Theory*, 22, 644-654.

[2] Miller, V.S. (1986). Use of elliptic curves in cryptography. *Advances in Cryptology - Proceedings of CRYPTO'85*, Lecture Notes in Computer Science, 218, 417-426. Springer-Verlag, New York.

[3] Koblitz, N. (1987). Elliptic curve cryptosystems. *Mathematics of Computation*, 48, 203-209.

[4] Silverman, J. (1986). *The Arithmetic of Elliptic Curves*. Springer-Verlag, New York.

[5] Hankerson, D., Menezes, A. J., Vanstone, S. (2003). *Guide to Elliptic Curve Cryptography*. Springer-Verlag, New York.

[6] FIPS 186-2 (2000). Digital Signature Standard (DSS). Federal Information Processing Standards Publication, 186-2. National Institute of Standards and Technology.

[7] Abdalla, M., Bellare, M., Rogaway, P. (1998). DHAES: An encryption scheme based on the Diffie-Hellman problem. Submission to IEEE P1363a.

[8] SEC 1 (2009). Elliptic Curve Cryptography, version 2. Standards for Efficient Cryptography Group. Certicom. http://www.secg.org/collateral/sec1.pdf.

[9] ISO/IEC 18033-2 (2006). Information technology – Security techniques – Encryption algorithms – Part 2: Asymmetric ciphers. International Organization for Standardization/International Electrotechnical Commission.

[10] Mall, D., Zhong, Q. (2008). Open source is not enough. Attacking the EC-package of Bouncycastle version 1.x_132. *Cryptology ePrint Archive, report 2008/113*.

Information about author(s):

Víctor Gayoso Martínez obtained his degree as Telecommunication Engineer from the Polytechnic University of Madrid in 2002. Since then, he has been working in topics related to smart cards, Java technology and public key cryptography.

Luis Hernández Encinas obtained his Ph.D. in Mathematics from the University of Salamanca, in 1992. He is a researcher at the Department of Information Processing and Coding, Spanish Council for Scientific Research (CSIC). His current research interests include cryptography, algebraic curve cryptosystems, image processing and number theory.

Carmen Sánchez Ávila received the Ph.D. degree in Mathematical Sciences from the Polytechnic University of Madrid in 1993. At present she is Professor in the Department of Applied Mathematics, where se has been teaching different undergraduate courses as well as graduate courses in Biometric and Cryptography.