Java Card implementation of the Elliptic Curve Integrated Encryption Scheme using prime and binary finite fields

V. Gayoso Martínez¹, L. Hernández Encinas², and C. Sánchez Ávila³

¹ Universidad Francisco de Vitoria, Pozuelo de Alarcón (Madrid), Spain
`v.gayoso.prof@ufv.es`

² Department of Information Processing and Coding, Applied Physics Institute, CSIC, Madrid, Spain
`luis@iec.csic.es`

³ Department of Applied Mathematics to Information Technologies, Polytechnic University, Madrid, Spain
`carmen.sanchez.avila@upm.es`

Abstract. Elliptic Curve Cryptography (ECC) can be considered an approach to public-key cryptography based on the arithmetic of elliptic curves and the Elliptic Curve Discrete Logarithm Problem (ECDLP). Regarding encryption, the best-known scheme based on ECC is the Elliptic Curve Integrated Encryption Scheme (ECIES), included in standards from ANSI, IEEE, and also ISO/IEC. In the present work, we provide a comparison of two Java Card implementations of ECIES that we have developed using prime and binary fields, respectively.

Keywords: Java Card, elliptic curves, public key encryption schemes

1 Introduction

In the current world, cryptography is essential for the protection of data and communication systems. Whitfield Diffie and Martin Hellman sparked off a revolution when they introduced the concept of public-key cryptography in 1976. Since that year, a great number of cryptosystems have been proposed, though many have been proved to be unsuitable for commercial purposes due to vulnerabilities in their designs or to their high complexity. The best-known successful public-key cryptosystem is RSA [1, 2], which was the first algorithm known to be suitable for signing as well as encrypting data.

In 1985, Victor Miller [3] and Neil Koblitz [4] independently suggested the usage of elliptic curves defined over finite fields in cryptography [5]. In comparison with other public-key cryptosystems, Elliptic Curve Cryptography (ECC) uses significantly shorter keys. The reason for this fact is related to the hardness of the ECDLP, which is considered to be more difficult to solve than the Integer Factorization Problem (IFP) used by RSA or the Discrete Logarithm Problem (DLP) which is the basis of the ElGamal encryption scheme [6].

Smart cards are small, portable, tamper-resistant devices offering users convenient storage and processing capabilities, and as such they play a prominent role in providing the required security level in banking transactions, the GSM and UMTS cellular systems or pay TV environments, to put only a few examples. Smart cards are amenable to cryptographic implementations, as they contain multiple software and hardware countermeasures designed to protect sensible information such as the keying material [7].

The two most widely used smart card platforms are Java Card [8], developed by Sun with the support from several leading smart card providers during the 1990s, and MULTOS (a multi-application operating system controlled by the MULTOS Consortium) [9]. Though both platforms implement cryptographic capabilities, the public-key cryptography capabilities provided by Java Card are more complete, specially regarding ECC functionality.

In the present work, we describe two implementations, using prime and binary fields, of the best known encryption scheme using elliptic curves, the Elliptic Curve Integrated Encryption Scheme (ECIES). To our knowledge, there are no smart card implementations of this encryption scheme, so this is the first time that ECIES is implemented, and its performance is analysed, in Java Cards. We also provide a performance comparison of both implementations using different combinations of plaintext lengths and key sizes.

This paper is organized as follows: Section 2 presents a brief introduction to ECC and ECIES. Section 3 includes a summary of the ECIES support in Java Card. Section 4 provides the main characteristics of the smart cards used in the tests and offers some details about the applet development phase. Finally, in Section 5 we present the experimental results of the tests along with the most important findings and conclusions.

2 Elliptic Curve Cryptography and ECIES

An elliptic curve E defined over a finite field $\mathbb{F}$ is a plane non-singular cubic curve with at least a rational point [10]. In practice, generic elliptic curves are managed using the following equation, known as the Weierstrass equation in non-homogeneous form [11], where the elements $a_1, a_2, a_3, a_4, a_6 \in \mathbb{F}$ and $\Delta \neq 0$, being Δ the discriminant of the curve E [12]:

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6. \quad (1)$$

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

When working with finite fields, it is possible to obtain simplified versions of the Weierstrass equation. If the finite field is a prime field, i.e. $\mathbb{F} = \mathbb{F}_p$, where $p > 3$ is a prime number, the equation defining the elliptic curve becomes:

$$y^2 = x^3 + ax + b. \quad (2)$$

On the other hand, if the finite field is a binary field, i.e. $\mathbb{F} = \mathbb{F}_{2^m}$, where m is a positive integer number, then the equation of the elliptic curve is:

$$y^2 + xy = x^3 + ax^2 + b. \quad (3)$$

Typical applications in security are key exchange, digital signatures and data encryption. For those three applications, there are lots of standards, some of them implemented through ECC. The best known ECC schemes and protocols are the Elliptic Curve Diffie Hellman (ECDH), a key agreement protocol [3, 4]; the Elliptic Curve Digital Signature Algorithm (ECDSA), equivalent to the DSA algorithm [13]; and the Elliptic Curve Integrated Encryption Scheme, the most extended ECC encryption scheme, defined in ANSI X9.63 [14], IEEE 1363a [15], and ISO/IEC 18033-2 [16].

As its name properly indicates, ECIES is an integrated encryption scheme which uses the following functions:

- Key Agreement (KA): Function used for the generation of a shared secret by two parties.
- Key Derivation Function (KDF): Mechanism that produces a set of keys from keying material and some optional parameters.
- Encryption (ENC): Symmetric encryption algorithm.
- Message Authentication Code (MAC): The output of the MAC function is the data used to authenticate a message.
- Hash (HASH): Digest function, used within the KDF and the MAC function.

3 ECIES support in Java Card

Java Card is a framework for programming and executing applications in smart cards developed by Sun and several smart card providers. Java Card version 2.1 already presented some cryptographic capabilities (e.g. DES, RSA, SHA-1, etc.), but the support for ECC was not included until version 2.2.

Even though ECDH and ECDSA are available in several versions of Java Card, and thus a programmer can use both ECC schemes by calling the proper methods of the Java Card API, this is not the case for ECIES. In fact, not only ECIES is not directly implemented as a single primitive in Java Card, but some of the functionality needed (e.g. the KDF and some MAC functions) must be developed by the programmer. To our knowledge, no ECIES implementation has been developed in Java Card prior to ours.

Java Card 2.2.1 implements the following functionality related to ECIES:

- KA function: Both Diffie-Hellman (DH) and the related Diffie-Hellman with cofactor (DHC), with the peculiarity that, instead of the product of the sender's ephemeral private key and the recipient's public key, both functions provide the SHA-1 output of that result.
- ENC function: AES with key length 128, 192, and 256 bits in modes CBC and ECB (in both cases without padding).

- HASH function: SHA-1.
- ECC key length: 113, 131, 163, and 193 bits in binary fields, and 112, 128, 160, and 192 bits in prime fields.

In comparison to the previous version, Java Card 2.2.2 included new MAC and hash functions. More precisely, in this version programmers were able to use the following functions that were not previously available: SHA-256, SHA-384, SHA-512, HMAC-SHA-1, HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512.

4 Smart cards used in the tests and applet development

The smart card models used in the tests presented in this contribution, namely JCOP 41 and JCOP J3A, have been kindly provided by NXP Semiconductors. All the information presented in this section is publicly available at the NXP web site [17–19].

Table 1 summarizes the main characteristics of JCOP 41 [17, 18] and JCOP J3A cards [17, 19]:

Table 1. JCOP 41 and JCOP J3A features

	JCOP 41	JCOP J3A
Hardware module	P5CT072	P5CD080
CPU	Secure MX51	Secure MX51
Operating system	JCOP 2.2.1	JCOP 2.4.1
Java Card version	2.2.1	2.2.2
GlobalPlatform version	2.1.1	2.1.1
ROM	160 Kbytes	200 Kbytes
EEPROM	72 Kbytes	80 Kbytes
RAM	4608 bytes	6144 bytes
Cryptographic coprocessors	3DES and PKI	3DES, AES, and PKI

It must be clarified at this point that the JCOP 41 product implements ECC for curves defined over $\mathbb{F}_{2^m}$, while the JCOP J3A model allows to use elliptic curves defined over $\mathbb{F}_p$. This is the reason why the available key sizes are not the same in those cards (131, 163, and 193 bits in JCOP 41 cards, and 128, 160, and 192 in JCOP J3A cards). For comparison purposes, in practice we will treat keys of similar lengths as if they were of the same size, as the difference in bits is not significant (131 vs 128, 163 vs 160, and 193 vs 192).

The applets for the JCOP 41 and JCOP J3A products have been developed by us using the Eclipse software (version 3.2) and the JCOP Tools plug-in for Eclipse kindly provided by NXP Semiconductors. This plug-in allows to easily compile, download and install Java Card applets in the JCOP smart cards.

With the goal of producing a valid comparison, we decided to implement the same ECIES functionality in both platforms, with the sole exception of the finite

fields and the key lengths used (binary finite fields with key lengths of 131, 163, and 193 bits in JCOP 41 cards, and prime finite fields with key lengths of 128, 160, and 192 in JCOP J3A cards).

The most important common features of the applets for both platforms are the following:

- KA function: DH, where the output of the function is the SHA-1 value of the shared secret.
- KDF function: KDF1 [16].
- ENC algorithm: 3DES with two keys, in CBC mode, and without padding.
- MAC function: HMAC-SHA-1 [20].
- HASH function: SHA-1.

As the maximum length for the APDUs transmitted to and from the smart cards using the T=0 protocol is 255 bytes [21], the plaintexts managed in the tests performed in this contribution have a maximum size of 160 bytes, so both the cryptogram provided by the smart card during the encryption process and the cryptogram sent to the smart card in the decryption process can be inserted in a single APDU.

The application code size in EEPROM is 4940 bytes for the JCOP 41 card and 4328 bytes in the case of the JCOP J3A card, as informed by Eclipse. The reason for this difference is that the JCOP 41 implementation manages keys of 4 possible sizes (113, 131, 163, and 193 bits) whilst the JCOP J3A manages only three types of keys (128, 160, and 192 bits). Regarding the keys of 113 bits available at the JCOP 41 cards, though implemented, we decided not to use them in this comparison, as that key size does not have a counterpart of a similar length in the JCOP J3A model.

Finally, it is worth mentioning that the elliptic curves implemented by the JCOP cards are standard curves published by SECG (sect131r1, sect193r1, secp128r1, and secp160r1), WAP Forum (c2pnb163v1), and ANSI (P-192).

5 Experimental results and conclusions

In order to be able to measure the encryption and decryption time, we developed a Java application using the Java Smartcard I/O API [22], available since Java SE 6, and the `System.nanoTime()` method included in that API, which returns the current value in nanoseconds of the most precise available system timer.

Using that timing function, the starting time has been taken exactly before sending the command APDU with the encryption/decryption request to the smart card, while the finishing time for each measurement has been obtained just after receiving the response APDU with the output of the encryption/decryption process from the smart card.

All the tests have been performed 20 times for each combination of message length (64, 96, 128, and 160 bytes) and key length (131, 163, and 193 bits when using the prime curves of JCOP 41, and 128, 160, and 192 bits in the case of the prime curves included in JCOP J3A).

The data included in Table 2 show the mean value of the 20 tests performed for every encryption and decryption combination in JCOP 41 and JCOP J3A cards. Taking into account all the tests, the maximum standard deviation computed is 0.81 milliseconds, which provides useful information about the stability of the smart card performance. Due to this stability, we decided that it was not necessary to perform additional tests, as the measures taken are concentrated around each average value and the CV in the worst-case scenario is 0.00286.

The results depend on the Java Card applet and the particular software optimizations applied (size of the arrays, variable usage and sharing, etc.), which means that a different applet could produce different values.

Table 2. JCOP 41 encryption and decryption time in milliseconds

			Encryption				Decryption			
			Message length (bytes)				Message length (bytes)			
			64	96	128	160	64	96	128	160
JCOP 41	Key length (bits)	131	433.79	463.97	498.25	531.29	279.02	280.32	296.01	297.87
		163	461.07	491.89	525.60	554.59	291.71	293.15	309.11	310.76
		193	499.28	529.95	563.75	593.98	311.19	313.08	328.78	331.00
JCOP J3A	Key length (bits)	128	489.58	498.19	517.37	520.19	436.72	439.72	453.35	455.42
		160	532.51	535.03	549.06	551.17	459.89	462.83	476.58	479.01
		192	597.85	601.28	615.44	617.52	523.09	525.75	539.93	542.82

Figures 1 and 2 present the performance of JCOP 41 and JCOP J3A when encrypting and decrypting messages of 64 and 160 bytes, respectively. Those figures include the processing results for key lengths of 131, 163, and 193 bits (JCOP 41), and 128, 160, and 192 bits (JCOP J3A).

After reviewing the data, it can be stated that the encryption time is relatively similar in JCOP 41 and JCOP J3A cards. However, it seems that the JCOP J3A implementation is more sensible to an increase in the key size than the JCOP 41 applet when maintaining the length of the plaintext.

Observing the decryption time, it is clear that there is an important gap (approximately 150-200 milliseconds, depending on the key length and message size) regarding the decryption process in both smart cards. Given that the applet functionality is basically the same (all the variables are located in RAM, the arrays have the same length in all the cases, etc.) this difference can be only explained due to a difference in the DES cryptoprocessor used in those models or to the countermeasures put in place in each card to protect them against side channel attacks.

In both smart cards, the decryption process is faster than the encryption procedure, basically because in the encryption process there is one step (the generation of the ephemeral key pair) which is not performed during the decryption process.

The quite noticeable increase in the encryption processing time for the JCOP 41 cards can be also visualized in Figure 1, where it can be seen how the en-

Fig. 1. Processing time in JCOP 41 and JCOP J3A for messages of 64 bytes.

Fig. 2. Processing time in JCOP 41 and JCOP J3A for messages of 160 bytes.

ryption time gap existing between JCOP 41 and JCOP J3A almost disappears when passing from plaintexts of 64 bytes to clear messages of 160 bytes. This result suggests that, when encrypting plaintexts of more than 160 bytes, JCOP 41 cards will be slower than the JCOP J3A model using the same configuration.

Another consequence of the increase in the relative difference of the encryption and decryption time in the JCOP 41 cards is that, when encrypting messages of 160 bytes, the time needed to encrypt the plaintext nearly doubles the time needed to decrypt the cryptogram.

Acknowledgement. This work has been partially supported by the “Fundación Memoria D. Samuel Solórzano Barruso” under the Project FS/7-2010.

References

1. Rivest, R., Shamir, A., Adleman L.: A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM.* 26, 96–99 (1983)
2. Menezes, A.J., Oorschot, P., Vanstone, S.: *Handbook of applied cryptography*. CRC Press, Florida (1996)
3. Miller, V.S.: Use of elliptic curves in cryptography. In: Williams, H.C. (ed.) *Advances in Cryptology-CRYPTO’85*. LNCS, vol. 218, pp. 417–426. Springer, Heidelberg (1986)
4. Koblitz, N.: Elliptic curve cryptosystems. *Math. Comput.* 48, 203–209 (1987)
5. Menezes, A.J.: *Elliptic curve public key cryptosystems*. Kluwer Academic Publishers, Boston (1993)
6. ElGamal, T.: A public-key cryptosystem and a signature scheme based on discrete logarithm. *IEEE Trans. Inform. Theory* 31, 469–472 (1985)
7. Preneel, B.: A survey of recent developments in cryptographic algorithms for smart cards. *Comput. Netw.* 51, 2223–2233 (2007)
8. Oracle Corporation. Java Card Technology, <http://java.sun.com/javacard/>
9. MAOSCO Limited. MULTOS smart card technology, <http://www.multos.com>
10. Cohen, H., Frey, G.: *Handbook of elliptic and hyperelliptic curve cryptography*. Chapman & Hall/CRC, Florida (2006)
11. Silverman, J.H.: *The arithmetic of elliptic curves*. Springer-Verlag, New York (1986)
12. Hankerson, D., Menezes, A.J., Vanstone, S.: *Guide to elliptic curve cryptography*. Springer-Verlag, New York (2004)
13. NIST FIPS 186-3. Digital Signature Standard (DSS). National Institute of Standards and Technology (2009)
14. ANSI X9.63. Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography. American National Standards Institute (2001)
15. IEEE 1363a. Standard Specifications for Public Key Cryptography - Amendment 1: Additional Techniques. Institute of Electrical and Electronics Engineers (2004)
16. ISO/IEC 18033-2. Information Technology – Security Techniques – Encryption Algorithms – Part 2: Asymmetric Ciphers. International Organization for Standardization / International Electrotechnical Commission (2006)
17. NXP Semiconductors. Smart Solutions for Smart Services, http://www.nxp.com/acrobat_download2/literature/9397/75016728.pdf
18. NXP Semiconductors. P5CT072 - Secure Dual Interface PKI Smart Card Controller (2004), http://www.nxp.com/acrobat_download2/other/identification/sfs085512.pdf
19. NXP Semiconductors. P5Cx012/02x/40/73/80/144 family - Secure Dual Interface and Contact PKI Smart Card Controller (2008), http://www.nxp.com/documents/data_sheet/P5CX012_02X_40_73_80_144_FAM_SDS.pdf
20. IETF RFC 2104. HMAC: Keyed Hashing for Message Authentication. Internet Engineering Task Force (1997), <http://www.ietf.org/rfc/rfc2104>
21. Rankl, W., Effing, W.: *Smart card handbook*. John Wiley & Sons, West Sussex, England (2003)
22. Oracle Corporation. Java smart card I/O API, <http://jcp.org/en/jsr/detail?id=268>