Security and Practical Considerations When Implementing the Elliptic Curve Integrated Encryption Scheme

Security and Practical Considerations when Implementing the Elliptic Curve Integrated Encryption Scheme

V. Gayoso Martínez, L. Hernández Encinas
Institute of Physical and Information Technologies (ITEFI)
Spanish National Research Council (CSIC), Spain
{victor.gayoso,luis}@iec.csic.es

A. Queiruga Dios
E.T.S.I.I. Béjar
University of Salamanca, Spain
queirugadios@usal.es

Abstract

The most popular encryption scheme based on elliptic curves is the Elliptic Curve Integrated Encryption Scheme (ECIES), which is included in ANSI X9.63, IEEE 1363a, ISO/IEC 18033-2, and SECG SEC1. These standards offer many ECIES options, not always compatible, making it difficult to decide what parameters and cryptographic elements to use in a specific deployment scenario. In this work, we show that a secure and practical implementation of ECIES can only be compatible with two of the four previously mentioned standards. In addition to that, we provide the list of functions and options that must be used in such an implementation. Finally, we present the results obtained when testing this ECIES version implemented as a Java application, which allows us to produce some comments and remarks about the performance and feasibility of our proposed solution.

Keywords data encryption, elliptic curves, Java, public key cryptography, standards

1 Introduction

The aim of this contribution is to present ECIES and to identify the peculiarities of the different versions of that encryption scheme that have been standardised in

ANSI X9.63 [4], IEEE 1363a [25], ISO/IEC 18033-2 [27], and SECG SEC1 [59]. Those versions offer a large number of implementation options, making it impossible to define a secure version of ECIES fully compatible with all the standards. In the present work, we have analysed the most relevant options of ECIES from a security and performance point of view and, as a result of that research, we show that a practical implementation of ECIES can only be compatible with two standards at the same time. Based on that knowledge, together with the information provided by the most recent attacks against this cryptosystem, we propose a set of functions and security recommendations that should be taken into account by any developer who intends to implement ECIES in a secure and efficient way. After that, we offer some results of our Java implementation of ECIES and a comparison with a similar scheme based on RSA and AES, which allows us to share some conclusions about the feasibility of the solution that we propose.

In [19] and [20], the authors provided a brief functional description of ECIES and an analysis of the different versions of this encryption scheme included in several standards, along with a list of the related functions used by ECIES (with a special focus on the ECIES functions available in Java Card in [20]). In comparison, in this contribution we offer a much more detailed description of ECIES (including a complete explanation of all the steps that must be performed in order to carry out the encryption and decryption operations); a presentation of the most important attacks against ECIES (which are necessary for understanding our recommendations); a security analysis of the different functions and special options allowed in each standard; a recommended set of functions and parameters based not only on their security, but also on their availability across the different standards; the experimental results obtained with a Java implementation of the version of ECIES which includes our proposed configuration; and a comparison with a similar scheme devised by us based on RSA and AES.

This paper is organized as follows: Section 2 presents a brief introduction to Elliptic Curve Cryptography (ECC). Section 3 describes in detail ECIES and the steps that must be performed during its encryption and decryption operation. Section 4 enumerates the most important attacks on ECIES. In Section 5, we offer a comparison of the ECIES allowed functions contained in the aforementioned standards. Section 6 explains additional configuration options for ECIES. Section 7 summarizes our proposed configuration for the encryption scheme. Section 8 provides the results obtained with our Java implementation of ECIES and a comparison with a similar schemed based on RSA and AES designed by us. Finally, Section 9 includes the conclusions about the feasibility of the ECIES version proposed by us.

2 Elliptic Curve Cryptography

It is well known that Miller [35] and Koblitz [30] independently proposed a cryptosystem based on elliptic curves, whose security relies on the Elliptic Curve Discrete Logarithm Problem (ECDLP). This problem can be defined as follows: given an elliptic curve E defined over a finite field $\mathbb{F}_q$ of q elements, a point G on the curve $E(\mathbb{F}_q)$ of order n, and a point P on the same curve, find the integer $k \in [0, n-1]$ such that $P = k \cdot G$ [24].

So far, no algorithm is known that solves the ECDLP in an efficient way, and it is supposed that this problem is more difficult to solve than other mathematical problems used in Cryptography, such as the Integer Factorization Problem or the Discrete Logarithm Problem [14]. This characteristic makes ECC a particularly well-suited option for devices with limited resources such as smart cards and some mobile devices [18, 28, 53].

In order to clarify the notation, we will briefly present some basic definitions and properties of elliptic curves. An elliptic curve over a finite field is defined by the following general Weierstrass equation [34]:

$$E(\mathbb{F}_q) : y^2 + a_1\, x\, y + a_3\, y = x^3 + a_2\, x^2 + a_4\, x + a_6, \tag{1}$$

where $a_1, a_2, a_3, a_4, a_6 \in \mathbb{F}_q$ and $\Delta \neq 0$, Δ being the discriminant of the curve.

In practice, instead of the general Weierstrass equation, two short Weierstrass forms that depend on the characteristic of the finite field $\mathbb{F}_q$ are typically used:

- If the finite field has p elements, where $p > 3$ is a prime number, then $\mathbb{F}_q = \mathbb{F}_p$, and the equation (1) is reduced to

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

- If the finite field has 2^m elements, then $\mathbb{F}_q = \mathbb{F}_{2^m}$, and the equation (1) can be written as follows:

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

The set of parameters to be used in any ECC implementation depends on the underlying finite field. When the field is $\mathbb{F}_p$, the set of parameters that define the curve is $\mathcal{P} = (p, a, b, G, n, h)$, whereas if the finite field is $\mathbb{F}_{2^m}$, the set of parameters is $\mathcal{P} = (m, f(x), a, b, G, n, h)$. The meaning of each element in both sets is the following:

- p is the prime number that characterizes the finite field $\mathbb{F}_p$.
- m is the integer number specifying the finite field $\mathbb{F}_{2^m}$.
- $f(x)$ is the irreducible polynomial of degree m defining $\mathbb{F}_{2^m}$.
- a and b are the elements of the finite field $\mathbb{F}_q$ taking part in the equations (2) and (3).
- G is the point of the curve that will be used as a generator of the points that belong to a cyclic subgroup of the curve.
- n is the prime number whose value represents the order of the point G.
- h is the cofactor of the curve, computed as $h = \#E(\mathbb{F}_q)/n$, where $\#E(\mathbb{F}_q)$ is the number of points on the curve.

3 Elliptic Curve Integrated Encryption Scheme

3.1 The road to ECIES

The Discrete Logarithm Augmented Encryption Scheme (DLAES) was introduced in [7], and it was later improved in [1] and [2], though by then it was renamed as the Diffie-Hellman Integrated Encryption Scheme (DHIES) in order to avoid misunderstandings with the Advanced Encryption Standard (AES) [36].

DHIES is an extended version of ElGamal encryption scheme, using elliptic curves in an integrated scheme that includes public key operations, symmetric encryption algorithms, Message Authentication Code (MAC) functions, and hash computations. This integrated scheme is secure against chosen ciphertext attacks without having to increase the number of elliptic curve operations (which are the slowest operations involved) or the key length [2].

DHIES represents the kernel of ECIES, which is the generic term used to define the best known encryption scheme using elliptic curves.

3.2 Functional components of ECIES

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

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

Figures 1 and 2 show graphic descriptions of the ECIES encryption and decryption procedures, including the elements and functions involved in both processes.

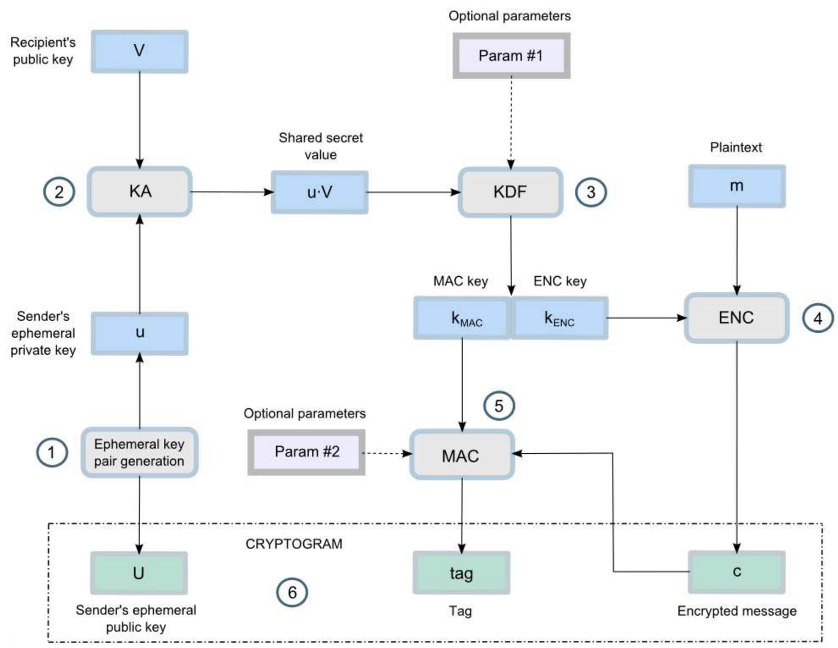


Figure 1: ECIES encryption process.

In order to describe the steps that must be taken to encrypt a plaintext, we will follow the tradition and will assume that Alice wants to send a message to Bob. In that scenario, Alice's ephemeral private and public keys will be represented as

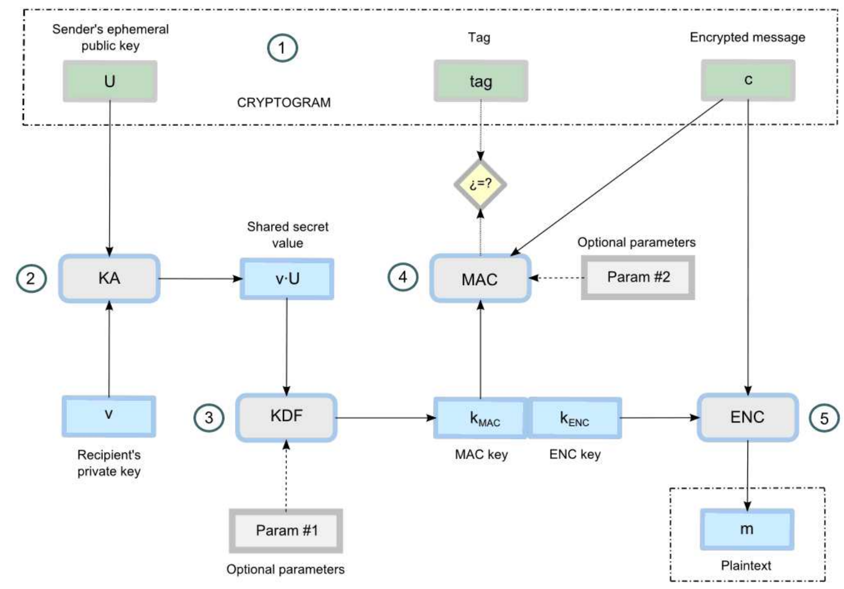


Figure 2: ECIES decryption process.

u and U, respectively. Similarly, we will refer to Bob's private and public keys as v and V, respectively. The steps that Alice must complete in order to encrypt a plaintext are the following:

1. Create a pair of ephemeral keys. The ephemeral private key is u, an integer modulo n chosen at random, whilst the ephemeral public key is $U = u \cdot G$.

2. Use the key agreement function, KA, in order to produce a shared secret value. If the Diffie-Hellman (DH) primitive is used, the secret value is the product of Alice's ephemeral private key and Bob's public key, $u \cdot V$. Alternatively, if the Diffie-Hellman with cofactor (DHC) is used [44], then the secret value is computed as the product of the cofactor, the sender's ephemeral private key, and the receiver's public key, $h \cdot u \cdot V$.

3. Take the shared secret value along with some optional parameters, identified as *Param* $\#1$, as input data for the key derivation function, denoted as KDF. The output of this function is the concatenation of the MAC key, k_{MAC}, and the encryption key, k_{ENC}.

4. Encrypt the plaintext, m, using the ENC symmetric algorithm and the encryption key, k_{ENC}. The ciphertext will be represented as c.

5. Use the selected MAC function, together with the encrypted message, the MAC key, and some optional parameters, identified as *Param* $\#2$, in order to produce a *tag*.

6. Take the ephemeral public key, the encrypted message, and the *tag*, and send the cryptogram consisting of those three elements to Bob. A simple method for sending that information is concatenating the three elements, so in that case the cryptogram would be represented as $(U||tag||c)$, where $||$ is the concatenation operator. It is important to note that the cryptogram defined in this way is not the same as the ciphertext, as in addition to the encrypted message, the cryptogram includes other two elements (the ephemeral public key and the tag).

Regarding the decryption process, the steps that Bob must perform in order to obtain the original message are the following:

1. Retrieve from the cryptogram the ephemeral public key U, the *tag*, and the encrypted message c, so he can manage those elements separately.

2. Use the retrieved ephemeral public key, U, and his own private key, v, to multiply both elements (optionally with the cofactor) in order to produce the shared secret value, as $u \cdot V = u \cdot v \cdot G = v \cdot u \cdot G = v \cdot U$ [24].

3. Produce the encryption and MAC keys by means of the KDF algorithm, the shared secret value, and the same optional parameters that Alice used before (*Param* $\#1$).

4. Compute the element *tag** using the MAC key k_{MAC}, the encrypted message c, and the same optional parameters used by Alice (*Param* $\#2$). After that, Bob must compare the *tag** value with the *tag* that he received as part of the cryptogram. If the values are different, the receiver must reject the cryptogram due to a failure in the MAC verification procedure.

5. Decrypt the ciphertext c using the symmetric ENC algorithm and k_{ENC}. At the end of the decryption process, Bob will be able to access the plaintext that Alice intended to send him.

4 Known attacks against ECIES

Although ECIES is a relatively new encryption scheme, it has been reviewed extensively by the research community. This review process has exposed a number of threats and attacks against ECIES, though fortunately those threats are either not practical or can be disabled with a proper configuration. The next subsections describe the most important theoretical and practical attacks on ECIES.

4.1 Benign malleability

Shoup proved that, if the ephemeral public key U is not included in the input to the KDF function, and only the x-coordinate of the shared secret is used in the KDF function, then ECIES is not secure against Adaptive Chosen Ciphertext Attacks (CCA2), making the scheme malleable [57]. More specifically, given a cryptogram $(U||c||tag)$, if the attacker replaces the elliptic curve point U by $-U$, then the KA function generates the shared secret $-u \cdot V$ instead of $u \cdot V$. But, taking into account that both points $u \cdot V$ and $-u \cdot V$ have the same x-coordinate, the input to the KDF function is the same in both cases, so from a valid cryptogram $(U||c||tag)$, the attacker is able to construct another valid cryptogram $(-U||c||tag)$.

In case of using the DHC primitive as the KA function, Shoup proved that it is also possible to create an attack making the scheme malleable [57]. To prevent this attack, an element whose order divides the cofactor h can be added to the elliptic curve point U. Another option to avoid this problem consists in using the DH primitive instead of the DHC version [59], as the DH primitive is allowed in the four standards analysed (see §5.1) and the cofactor variant can lead to interoperability problems [10, 59].

Shoup defined these type of problem as *benign malleability*, as so far no attack has been able to obtain relevant information using this threat. However, from a formal point of view, it is important to avoid these type of vulnerabilities (as Shoup stated in [57], "for public-key encryption schemes, it is widely agreed that the right notion of security for a scheme intended for general-purpose use is that of security against adaptive chosen ciphertext attack. This notion was introduced in [52], and implies other useful properties, like non-malleability").

4.2 Malleability when using the XOR function

Shoup also proved in [57] that the ECIES scheme could be malleable, but now in a malign way, when the XOR function is used in order to encrypt messages of variable length, which could give way to attacks of type CCA2. Some solutions to this problem are described below:

1. Establish a fixed length for all the plaintexts [57].
2. Fix the interpretation of the MAC and ENC keys obtained from the keying material which is the output of the KDF function, so those keys are always interpreted as $k_{MAC}||k_{ENC}$, as is suggested in [1], [57], and [60].
3. Forbid the usage of stream ciphers in ECIES, allowing only block ciphers, as recommended in [51] and [57].

As the main use case for ECIES consists in the encryption of text messages or binary files of arbitrary size, and in some cases the XOR key size could be really big, from a practical point of view it is recommended to use block ciphers.

Regarding the second solution proposed, as it is mentioned in §6.3, the interpretation $k_{MAC}||k_{ENC}$ is only allowed in IEEE 1363a, so we should discard this option if the goal is to develop a version of ECIES compatible with more than one standard.

Taking into account the previously stated considerations, we fully support the recommendation of not using XOR as an encryption algorithm in this encryption scheme.

4.3 Small subgroup attacks

This type of attack is possible when an opponent deliberately provides an invalid public key, where in this context a valid public key is an elliptic curve point which belongs to the elliptic curve selected by the users and that presents the arithmetic properties described in [24]. If the sender does not check whether the other party's public key is valid, an opponent would be able to provide as the public key an element of small order, with the goal to limit the range for the shared secret value or to obtain information about the sender's private key. The options available for the deactivation of this kind of attack are:

1. Use the DH primitive and check carefully the validity of the parameters and of the public key provided by the receiver (e.g., check that the order of the public key V is n) [59].

2. Use the DHC primitive. If the public key V belongs to a small subgroup, then the element $h \cdot u \cdot V$ will be equal to the point at infinity $\mathcal{O}$, a well known point for any curve [57, 59].

3. Replace the shared secret by the hash code of the secret value as the input to the KDF function [25].

In a typical scenario, the validity check on the public key V would be performed by the trusted third party issuing certificates, so this validity checking should not impact on the performance of ECIES.

Regarding the option of using the DHC primitive, as it was explained in §4.1, it faces the theoretical threat of a malleability attack, that being one of the reasons why most test vectors included in the standards do not use the DHC primitive [27, 58].

The usage of the hashed output is mentioned in IEEE 1363a, and thus it has been implemented in Java Card since its version 2.2 [50]. However, this feature is used in the test vectors of neither ISO/IEC 18033-2 [27] nor SECG GEC 2 [58], and Java Card 3.0 has added another operation mode in which the output of the KA function is not hashed [49].

From the three possible solutions, we have selected the first one for our proposal, as the second option is regarded as not fully interoperable [10, 59], and the third possibility is mentioned in only one standard. Besides, if the ephemeral key pair is generated randomly and is used only once, then no practical information that could be used in new encryption processes would be obtained by an attacker using this method.

5 Allowed functions in standard ECIES

Given the number of functions and options involved, the major problem when using ECIES is to determine the proper combination of functions and parameters to use. In the following sections we will present the allowed functions included in the different versions of ECIES, together with the recommendations that we propose based on security and performance criteria.

5.1 Key Agreement function (KA)

The Key Agreement function produces a secret value that can only be obtained by both sender and receiver. The two KA functions used in ECIES are DH and DHC, which were described in §3.2.

Both DH and DHC are allowed in the four standards analysed. In devices with limited resources, the DH primitive may be slightly faster as it implies only one scalar multiplication. Besides, another reason for using DH instead of DHC is mentioned in §4.1. Given both reasons, we propose using DH as the KA function.

5.2 Hash function (HASH)

Hash functions take as input a binary string of variable length and produce as a result a binary string of fixed length corresponding to the initial data. In the scope of ECIES, hash functions are used by other primitives (e.g. KDF or MAC). The hash functions mentioned in the standards where ECIES is included are SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 [43], RIPEMD-128, RIPEMD-160 [16], and WHIRLPOOL [26]. Table 1 presents the hash functions used in each standard.

ANSI X9.63	IEEE 1363a	ISO/IEC 18033-2	SECG SEC 1
SHA-1	SHA-1	SHA-1	SHA-1
SHA-224	SHA-256	SHA-256	SHA-224
SHA-256	SHA-384	SHA-384	SHA-256
SHA-384	SHA-512	SHA-512	SHA-384
SHA-512	RIPEMD-160	RIPEMD-128	SHA-512
		RIPEMD-160	
		WHIRLPOOL	

Table 1: Hash functions.

During the past years, several attacks on MD5, SHA-0, SHA-1, SHA-2 have been published [5, 9, 15, 29, 54, 62–64]. Due to these attacks, NIST held a workshop to consider the status of hash functions at the end of 2005. The conclusions of the workshop were first to initiate a rapid transition to the SHA-2 family of hash functions (NIST considers that the SHA-2 functions are much stronger than SHA-1, and that practical attacks are unlikely to appear at least during the next few years [38]), and second to set up a hash function competition, similar to the AES development and selection process, in order to select the new SHA-3 algorithm. On October 2012 NIST announced Keccak as the new SHA-3 hash algorithm [42]. As the decision on SHA-3 is very recent, so far none of the standards where ECIES is described has updated their specifications in order to include the winning algorithm.

Taking into account the level of scrutiny performed by the expert community, from a security point of view we suggest to use one of the algorithms of the SHA-2 family. If memory and bandwidth limitations are critical requirements in the deployment scenario (e.g. smart cards), then we recommend using SHA-256. In contrast, if memory and bandwidth are not critical elements, then we suggest using SHA-512. Another argument in favour of selecting SHA-512 over SHA-256 is its better performance on 64-bit architectures [23], which are the current trend in laptop and desktop computers.

5.3 Key Derivation Function (KDF)

Key Derivation Functions (KDF) are used to generate keying material from a shared secret and additionally from other optional elements. The key derivation functions allowed by the different versions of ECIES are ANSI-X9.63-KDF [4], NIST-800-56-Concatenation-KDF [44], KDF1, and KDF2 [27]. The KDF functions considered in each standard version of ECIES are presented in Table 2.

ANSI X9.63	IEEE 1363a	ISO/IEC 18033-2	SECG SEC 1
X9.63-KDF	X9.63-KDF	KDF1	X9.63-KDF
		KDF2	NIST-800-56

Table 2: KDF functions.

In order to perform the comparison, it must be taken into account that, if the parameter *SharedInfo* is not used in X.63-KDF, then this function is equivalent to KDF2.

So far, no specific threats have been discovered against the previously mentioned KDF functions, so theoretically any of them could be used in a secure implementation of ECIES. Given that the KDF2 algorithm (or, equivalently, X.63-KDF without *SharedInfo*) is allowed by all the standards, we recommend to use it as the KDF algorithm.

5.4 MAC code generation function (MAC)

MAC functions take as input a binary string and produce as output another binary string (known as the *tag*) related to the input and to certain optional parameters. A specific type of MAC function is the HMAC group of functions that use a hash primitive as part of the computations. The MAC functions allowed in the standards where ECIES is included are HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512 [37], HMAC-RIPEMD-128, HMAC-RIPEMD-160 [31], CMAC-AES-128, CMAC-AES-192, and CMAC-AES-256 [39]. Table 3 shows the allowed MAC functions in the four standards, where for presentation reasons the word HMAC has been abbreviated to simply H.

ANSI X9.63	IEEE 1363a	ISO/IEC 18033-2	SECG SEC 1
H-SHA-1	H-SHA-1	H-SHA-1	H-SHA-1
H-SHA-224	H-SHA-256	H-SHA-256	H-SHA-224
H-SHA-256	H-SHA-384	H-SHA-384	H-SHA-256
H-SHA-384	H-SHA-512	H-SHA-512	H-SHA-384
H-SHA-512	H-RIPEMD-160	H-RIPEMD-128	H-SHA-512
		H-RIPEMD-160	CMAC-AES-128
		H-WHIRLPOOL	CMAC-AES-192
			CMAC-AES-256

Table 3: MAC functions.

In the case of using one of the SHA-2 functions as the hashing algorithm, we recommend using one of MAC functions belonging to the HMAC-SHA-2 family. If the deployment scenario has memory and bandwidth limitations, we propose using HMAC-SHA-256, in line with what was stated in §5.2. Otherwise, our suggestion would be to use HMAC-SHA-512.

5.5 Symmetric encryption function (ENC)

Symmetric encryption functions use a secret key in order to encrypt the input information. The symmetric algorithms included in the different versions of ECIES are XOR, Triple DES [40], AES-128, AES-192, AES-256 [36], MISTY1 [45], CAST-128 [3], Camellia [6], and SEED [32]. The symmetric ciphers considered in the standards that include ECIES are shown in Table 4, where the related operation modes are CBC (Chain Block Mode) and CTR (Counter), and the term PKCS5 refers to the PKCS #5 padding mechanism.

As it is well known, the AES specification comprises three block ciphers, AES-128, AES-192, and AES-256, adopted from a larger collection originally

ANSI X9.63	IEEE 1363a	ISO/IEC 18033-2	SECG SEC 1
XOR	XOR	XOR	XOR
	TDES/CBC/PKCS5	XOR$^{\perp}$	TDES/CBC
	AES/CBC/PKCS5	TDES/CBC/PKCS5	AES/CBC
		AES/CBC/PKCS5	AES/CTR
		MISTY1/CBC/PKCS5	
		CAST-128/CBC/PKCS5	
		Camellia/CBC/PKCS5	
		SEED/CBC/PKCS5	

Table 4: Symmetric encryption functions.

published as Rijndael. Each of these ciphers has a 128-bit block size, with key sizes of 128, 192, and 256 bits, respectively. Given the latest attacks on AES [8, 11], it is commonly agreed that, until new attacks are published, AES-128 is relatively more secure than AES-192 and AES-256 [55].

Regarding the mode of operation, even though CTR offers some advantages (it does not require padding, its implementation is more efficient, etc.) [33], as it is only considered a valid operation mode in SECG SEC 1, in order to make the ECIES implementation compatible with at least two standards, the AES operation mode that should be implemented is CBC with PKCS #5 padding.

5.6 Summary of functions allowed in all the standards

As a summary of the information that has been previously presented, Table 5 includes all the cryptographic functions and algorithms allowed simultaneously in the four standard versions of ECIES cited along this document.

KA	HASH	KDF	ENC	MAC
DH	SHA-1	KDF2	XOR	HMAC-SHA-1
DHC	SHA-256			HMAC-SHA-256
	SHA-384			HMAC-SHA-384
	SHA-512			HMAC-SHA-512

Table 5: Common functions allowed in the standards.

Even though there are several combinations compatible with the four standards (e.g. DH, SHA-1, KDF2, XOR, and HMAC-SHA-1), all of them have in common the same encryption algorithm, which is the XOR function. As it is stated in §4.2, using the XOR function for encrypting messages of variable length weakens the security of ECIES and makes the scheme less practical compared to the case of using a block cipher with fixed key lengths, so we consider that in this particular case it is better to sacrifice full interoperability in favour of security and practicality.

For that reason, our recommendation is to use the following set of functions in order to obtain a secure version of ECIES: DH, SHA-512, KDF2, HMAC-SHA-512, and AES-128 in CBC mode with PKCS #5 padding.

6 Additional options

After reviewing the currently known attacks against ECIES and their countermeasures in §4, and addressing the topic of which are the most convenient functions and algorithms in §5, this section focuses on the additional implementation decisions that must be considered in order to develop a secure and efficient implementation of ECIES.

6.1 Point compression usage

Point compression is a technique used when converting an elliptic curve point into a byte string in which it is decided to include in the binary representation either the first coordinate of the point or both coordinates, in both cases together with one byte that identifies the format that has been selected. The point compression formats that can be used by developers, and that are defined in the four standards analysed, are the following:

1. *Uncompressed*: Both coordinates are taken into account. A header byte $0\mathrm{x}04$ is used to indicate that this is the format in use, so the byte string corresponding to the elliptic curve point $P = (x_P, y_P)$ would be $04||X_P||Y_P$, where X_P and Y_P are the binary representations as integers of the affine coordinates x_P and y_P.

2. *Compressed*: Only the first coordinate is used, which is signalled by using the header byte $0\mathrm{x}02$ or $0\mathrm{x}03$. The exact value of the header is decided based on some computations performed involving both coordinates, which allows the receiver to accurately generate the second coordinate, so for any elliptic

curve point only one compressed binary representation, either $02||X_P$ or $03||X_P$, is valid.

From a security standpoint, there is no difference in transmitting to the other party as part of the cryptogram the ephemeral public key U in either compressed or uncompressed form [51, 60]. However, even though Miller suggested compressing a public key to simply its first coordinate [35], there are several patents over this topic [17, 56, 61], so in order not to infringe any of those patents we recommend not to use point compression.

6.2 KDF input data

Independently of which of the KA functions is used (DH or DHC), developers face a variety of options regarding the information that will be taken as input in the KDF function. The elements that can be used to construct the input are the following:

1. The point obtained as the output of the KA function (i.e., $P = u \cdot V$ or $P' = h \cdot u \cdot V$), or just the first coordinate of that point (i.e., x_P or $x_{P'}$).

2. The hash output of the elements previously mentioned (i.e., P, P', x_P or $x_{P'}$).

3. The point that represents the ephemeral public key U, either compressed or uncompressed.

4. Additional parameters.

Table 6 shows the options allowed in each standard, where x_P is the first coordinate of $P = u \cdot V$, x_U is the first coordinate of U, and $P\#1$ represents the optional parameters identified as *Param* $\#1$ in §3.2. The concatenation of binary strings is represented with the usual symbol, $||$, whilst the fact that two parameters are used as input (but not in a concatenated form) is displayed using a comma. For the sake of clarity we have presented only the options related to x_P, though all the options are also available when using $x_{P'}$ instead of x_P.

With regards to the implications of using the whole point representing the shared secret or only the first coordinate as input to the KDF function (not to be confused with the discussion in §6.1 about the format of the ephemeral public key U included in the cryptogram), some authors, such as Stern [60], state that from

ANSI X9.63	IEEE 1363a	ISO/IEC 18033-2	SECG SEC 1
x_P	x_P	x_P	x_P
$x_P, P\#1$	$x_P, P\#1$	$U\|\|x_P$	$x_P, P\#1$
	$U\|\|x_P$	$x_U\|\|x_P$	
	$x_U\|\|x_P$		
	$U\|\|x_P, P\#1$		
	$x_U\|\|x_P, P\#1$		

Table 6: KDF input data options.

a security point of view there is no difference in using any of the two options. After reviewing the standards that include ECIES, most of them take only the first coordinate of the shared secret as input to the KDF function, so this seems to be the commonly accepted solution in practice in order to produce more efficient implementations.

Concerning the option of including the ephemeral public key of the sender as an input to the KDF function, as it has been mentioned in §4.1, using the public key U can help to prevent benign malleability, so we propose to use $U||x_P$ as input to the KDF function in the ECIES implementation. We are aware that this decision affects to the interoperability of ECIES, as it is only valid in IEEE 1363a and ISO/IEC 18033-2, but we think that, in this particular case, security is more important than interoperability.

Another alternative which preserves the security of ECIES is using x_P as the first input parameter and U as the second parameter, though not concatenated, so in this case the implementation would be compatible with the format $x_P, P\#1$ allowed in IEEE 1363a and SECG SEC 1. The security level obtained with both options is the same; the difference is the list of standards the ECIES implementation would be compatible with.

6.3 Keying material interpretation

Before obtaining the MAC and ENC keys from the output of the KDF function, users must decide which is the interpretation order of that output. The two options available are:

1. First, the MAC key; then, the ENC key ($k_{MAC}||k_{ENC}$).
2. First, the ENC key; then, the MAC key ($k_{ENC}||k_{MAC}$).

All the analysed standards allow the $k_{ENC}||k_{MAC}$ interpretation when using a block cipher as the symmetric encryption algorithm. Only IEEE 1363a permits using the $k_{MAC}||k_{ENC}$ interpretation, and strictly under specific circumstances (stream cipher, etc.). Based on that, $k_{ENC}||k_{MAC}$ is the option that must be chosen for any practical ECIES implementation.

6.4 MAC input data

The four standards allow using as input to the MAC function either the encrypted message or the encrypted message concatenated with the optional parameters identified as *Param* $\#2$ in §3.2.

From a security standpoint, ECIES is slightly strengthened if sender and receiver share the content of this parameter, even if it is just a short passphrase not strong enough to constitute a secure secret key (e.g. 1234). However, it must be taken into account that this feature may not always be possible to use, for example when sending a message to a user with whom no prior contact has been made, so we think that using this option cannot be enforced as a general rule.

6.5 Dynamic selection of parameters and functions

Even though some authors consider it to be of the essence for the security of ECIES to use the same set of parameters and the same KA, KDF, ENC, and MAC functions during the whole life cycle of a specific key pair [57], and in IEEE 1363a this procedure is recommended, in practice there is no consensus among the experts about the risk that a change of parameters and functions would imply [51].

However, as the requirement made in [57] does not seem to have negative implications, we adhere to the recommendation of changing neither the parameters nor the functions during the life cycle of a given key pair.

6.6 Elliptic curve generation procedure

When working with elliptic curve protocols, an important topic is the selection of the specific curve to use. Even though elliptic curve generation procedures are defined in standards from ANSI, IEEE, and ISO/IEC, it is usually the case that the elliptic curve parameters are offered to the reader without a complete and verifiable generation process. Some of the most important limitations detected across the main cryptographic standards regarding this issue are the following [12]:

- 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 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 in 2005 it delivered the first version of a document entitled "ECC Brainpool standard curves and curve generation" [12], 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" [13].

The Brainpool specification includes the steps that must be performed in order to generate elliptic curves suitable for cryptographic purposes, and it also presents the functional and security requirements that must be taken into account when generating computationally efficient and secure elliptic curves.

Given that the Brainpool procedure is the most complete and publicly available procedure for generating elliptic curves, offering a range of key lengths for all possible security needs (from 160 to 512 bits), we suggest using elliptic curves generated by the Brainpool procedure.

Even though the Brainpool curves are different from the curves defined in other standards, selecting a working elliptic curve is independent of the encryption process itself, so it does not affect the interoperability of the ECIES implementation; it only affect its operation by the users.

7 ECIES configuration

Based on the comments and recommendations included in the previous sections, we summarize below the list of parameters, algorithms, and additional characteristics that allow implementing an efficient and secure version of ECIES, which is compliant with the version described in the standards IEEE 1363a and ISO/IEC 18033-2.

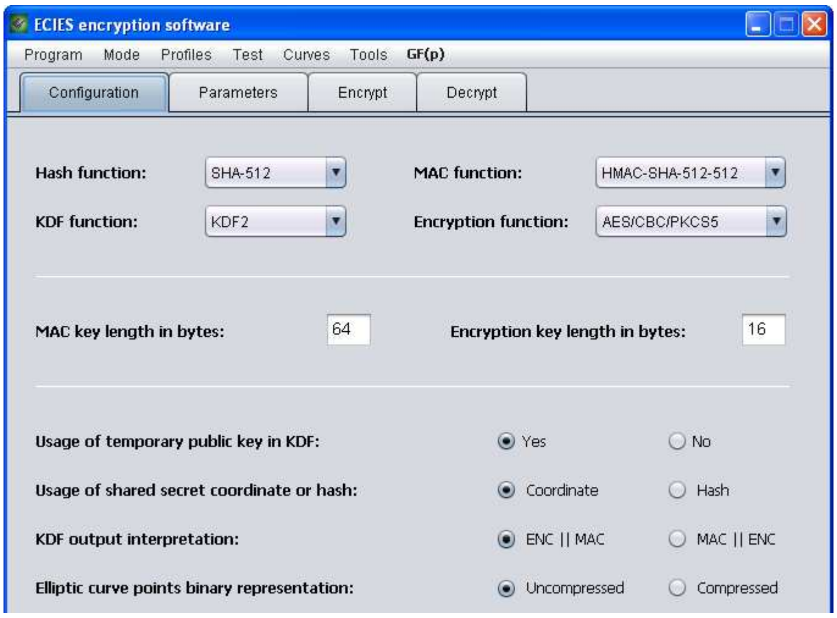
- KA: the DH function.
- Hash: SHA-512.
- KDF: KDF2.
- ENC: AES-128 in CBC mode with PKCS #5 padding.
- MAC: HMAC-SHA-512.
- Shared secret: use only the first coordinate (without hash).
- Input to the KDF function: ephemeral public key U concatenated to x_P.
- KDF output interpretation: $k_{ENC}||k_{MAC}$.
- Binary representation of U as part of the cryptogram: uncompressed.
- Selection of parameters and functions: static (for a given public key).

8 Experimental results

After proposing a configuration for ECIES, the next step consisted in conducting some tests in order to check the practicality and performance of the proposed solution. The software that was employed in the tests was developed by the authors as a Java application [47] that can be used to test different parameter combinations for ECIES [22]. Figure 3 displays part of its *Configuration* panel, where all the options can be selected.

The elliptic curves used in the tests belong to the group of curves published by Brainpool [12, 13, 21]. As for the specific curves employed in the tests, a representative of each available key length has been selected (160, 192, 224, 256, 320, 384, and 512 bits).

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. The Java software used for compiling and executing the application is Java Development Kit (JDK) 1.6 update 27 [48], with default parameters both for compiling and running the application.



Hash function: SHA-512

KDF function: KDF2

MAC function: HMAC-SHA-512-512

Encryption function: AES/CBC/PKCS5

MAC key length in bytes: 64

Encryption key length in bytes: 16

Usage of temporary public key in KDF: [✓ Yes] No

Usage of shared secret coordinate or hash: [✓ Coordinate] Hash

KDF output interpretation: [✓ ENC || MAC] MAC || ENC

Elliptic curve points binary representation: [✓ Uncompressed] Compressed

Figure 3: Java implementation of ECIES.

8.1 Message expansion factor

This section presents the message expansion factor (i.e., the ratio measured as the quotient of the cryptogram length and the clear message length) when using elliptic curves for the encryption of different messages.

Due to the nature of ECIES, the expansion in the cryptogram length (i.e., the difference between the length of the cryptogram and the length of the original message) depends on the following factors:

1. The finite field and associated field element representation.
2. The binary representation of the sender's ephemeral public key (either compressed or uncompressed).
3. The specific encryption function and the mode used.
4. The length of the MAC code.

Given that the message length in bytes used in the different tests of this section is, in all the test cases, a multiple of 16 (which produces an additional block of 16 bytes when encrypting the original message with the AES algorithm in CBC mode with PKCS #5 padding), the MAC code length is 64, and the elliptic curve point representation is uncompressed, then the message expansion factor δ is, for any curve defined over $\mathbb{F}_p$, given by the expression

$$\delta = 1 + 2 \cdot \left\lceil \frac{\log_2 p}{8} \right\rceil + 16 + 64 = 2 \cdot \left\lceil \frac{\log_2 p}{8} \right\rceil + 81,$$

where $\lceil n \rceil$ represents the ceiling function applied to the real number n. The decision of using messages whose length is a multiple of 16 has been taken in order to present the worst-case scenario in terms of expansion (if the message length is not a multiple of 16, then the amount of bytes that are needed for padding will be less than 16, and the expansion will also be smaller).

Table 7 shows the cryptogram length for different plaintext sizes (16, 32, 64, 128, 256, 512, and 1024 bytes) when using the Brainpool curves of 160, 192, 224, 256, 320, 384, and 512 bits. Besides, Table 8 and presents the ratio computed with those input values.

		Key length (bytes)						
		160	192	224	256	320	384	512
Plaintext (bytes)	16	137	145	153	161	177	193	225
	32	153	161	169	177	193	209	241
	64	185	193	201	209	225	241	273
	128	249	257	265	273	289	305	337
	256	377	385	393	401	417	433	465
	512	633	641	649	657	673	689	721
	1024	1145	1153	1161	1169	1185	1201	1233

Table 7: Cryptogram length in bytes.

As it can be observed in Figure 4, which displays the data offered in Table 8, the overload added by the encryption process, compared to the size of the ciphertext produced by a symmetric cipher such as AES, is irrelevant when encrypting more than 1 KByte of information. However, when encrypting a short message of less than 1 KByte, particularly when the data length is less than 64 bytes, the ratio is very high. This implies that integrated encryption schemes as ECIES are specially suited for the encryption of medium to large messages.

		Key length (bytes)						
		160	192	224	256	320	384	512
Plaintext (bytes)	16	8.56	9.06	9.56	10.06	11.06	12.06	14.06
	32	4.78	5.03	5.28	5.53	6.03	6.53	7.53
	64	2.89	3.02	3.14	3.27	3.52	3.77	4.27
	128	1.95	2.01	2.07	2.13	2.26	2.38	2.63
	256	1.47	1.5	1.54	1.57	1.63	1.69	1.82
	512	1.24	1.25	1.27	1.28	1.31	1.35	1.41
	1024	1.12	1.13	1.13	1.14	1.16	1.17	1.2

Table 8: Message expansion factor.

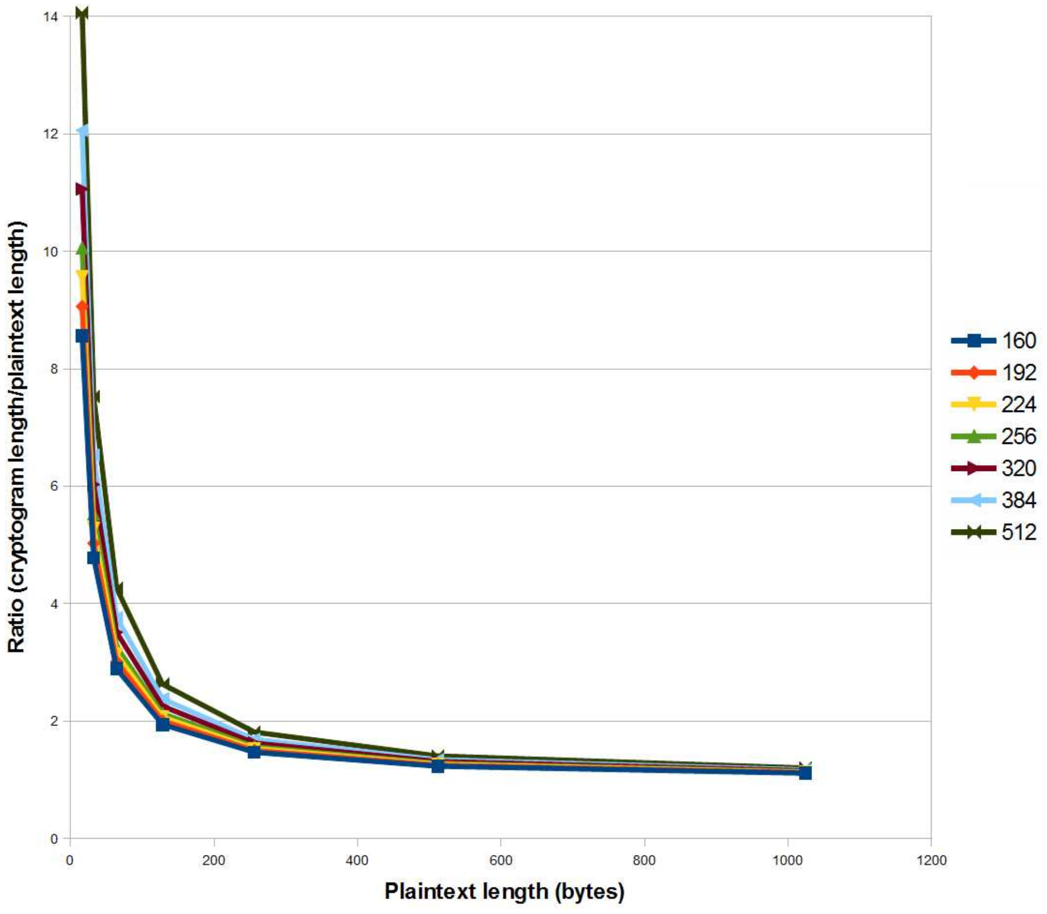


Figure 4: Message expansion factor for different elliptic curves and plaintexts.

8.2 Execution time

The details of the procedure used for obtaining the timing results of the encryption process in our PC implementation are the following:

1. For each encryption process, the same original message consisting of 1024 bytes has been used. For each decryption process, the output of the encryption procedure with each curve has been used as input.

2. The time displayed for each curve represents the average time of 10 different encryption/decryption processes.

3. The Java function used for obtaining the timing is `System.nanoTime()` [46].

4. In the encryption procedure, the start time has been taken exactly before the random generation of the ephemeral key U, after all the parameters and variables are loaded in memory. In comparison, the start time in the decryption procedure has been taken exactly before the $v \cdot U$ multiplication, as in that process there is no ephemeral key pair generation.

5. The finish time for each encryption procedure has been obtained just after the cryptogram is stored in memory as a hexadecimal string, and before any data is printed on screen (in order to avoid the delays produced when presenting the results in the graphic interface). Similarly, the finish time for each decryption procedure has been obtained after the recovered plaintext is stored in memory as a hexadecimal string.

Table 9 shows the average execution time for the Brainpool curves when encrypting and decrypting the same plaintext of 1024 bytes. As expected, the execution time is not linear, as ECC computations are more demanding as the key length is increased. In order to illustrate this comment, Figure 5 depicts the average encryption time (we have not included the decryption time in the figure as it is very similar to the encryption time and both charts would appear superimposed).

As observed in Table 9, the encryption and decryption time is practically the same. The reason for this behaviour is that both processes perform the same operations but for the ephemeral key generation, which is only performed in the encryption phase. The running time of the ephemeral key generation is negligible compared to the total execution time, so it is not a differentiation factor.

	Key length (bytes)						
	160	192	224	256	320	384	512
Encryption (ms)	7.59	10.33	13.91	17.14	26.90	40.95	81.70
Decryption (ms)	7.36	10.47	13.31	17.27	28.21	39.75	82.84

Table 9: Encryption and decryption time in milliseconds with ECIES.

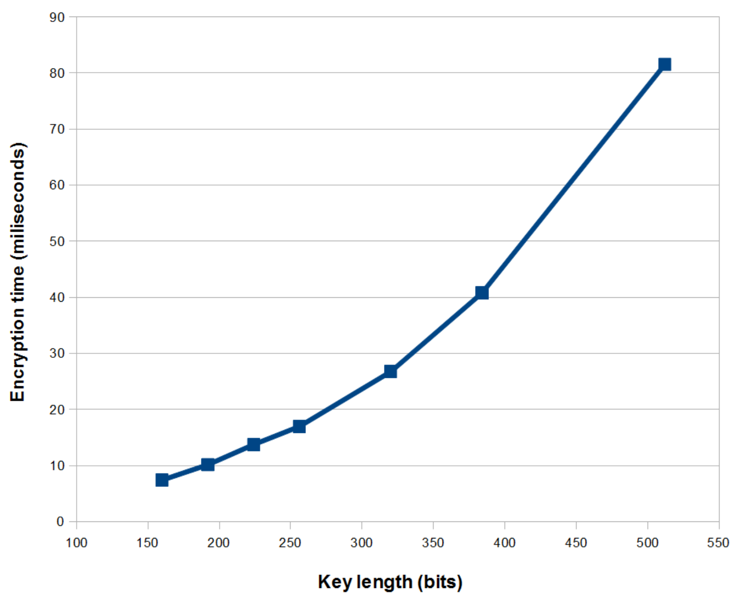


Figure 5: Encryption time for different elliptic curves.

8.3 Comparison with RSA

With the goal of completing a fair evaluation of the capabilities of ECIES, we have implemented in Java a hybrid encryption scheme based on RSA and AES comparable in its design to ECIES, so we could evaluate the cryptogram length and the execution time in both cases. Our RSA-AES scheme performs the following steps during the encryption phase:

1. It generates a random AES key of 16 bytes.
2. It encrypts a 1024-byte plaintext using AES-128 in CBC mode with PKCS #5 padding.
3. It encrypts the AES secret key using the receiver's RSA public key.
4. It computes an RSA signature using the hash function SHA-512 and the sender's private key, taking as input both the plaintext and the encrypted secret key. That signature acts as the MAC code employed in ECIES.
5. It outputs the cryptogram as the concatenation of the three previous elements: the data encrypted with the secret key, the encrypted secret key, and the signature.

In the decryption phase, our RSA-AES scheme completes the following steps:

1. It obtains the AES key using the receiver's RSA private key.
2. It retrieves the 1024-byte plaintext using AES-128 in CBC mode with PKCS #5 padding.
3. It validates the digital signature using the sender's RSA public key.

In all the RSA-AES tests we have encrypted the same plaintext of 1024 bytes used in some of the ECIES tests. The RSA key lengths managed are associated to ECC key lengths of similar cryptographic strength, where in this context the cryptographic strength must be interpreted as the security offered by a symmetric encryption algorithm with keys of n bits [41]. Table 10 includes the ECIES and RSA key lengths used in this comparison.

Table 11 shows the cryptogram length in bytes of ECIES and our RSA-AES scheme. The cryptogram length in the RSA-AES implementation can be computed as $1040 + 2 \cdot k_{len}$, where k_{len} represents the RSA key length. The symmetric

Security level	ECIES key length	RSA key length
80	160	1024
112	224	2048
128	256	3072
192	384	7680
256	512	15360

Table 10: ECIES and RSA comparable key lengths in bits (source: [41]).

encryption of 1024 bytes using AES in CBC mode with PKCS #5 padding produces an output of 1040 bytes, and the RSA encryption of the symmetric key and the RSA signature generate two blocks of the same size as the RSA key length used in each test. Given the data of Table 11, it can be observed that ECIES generates shorter cryptograms in all the instances, which is important when transmitting data through networks with small bandwidth or when the device producing the encrypted data has a limited communication channel (e.g. smart cards).

Security level	ECIES	RSA
80	1145	1296
112	1161	1552
128	1169	1808
192	1201	2960
256	1233	4880

Table 11: Cryptogram length in bytes with ECIES and RSA-AES.

Table 12 shows the encryption and decryption time in milliseconds of ECIES and our RSA-AES version. Even though for smaller security levels the scheme based on RSA and AES is faster than ECIES, for a security level of 128 bits the execution time is similar, and for the highest security levels ECIES is clearly faster than the RSA-AES scheme.

Figure 6 shows graphically the comparison between ECIES and RSA-AES regarding the encryption time (the decryption time is not included in the figure as it is very similar to the encryption time, making both charts to appear superimposed).

	Encryption		Decryption	
Security level	ECIES	RSA-AES	ECIES	RSA-AES
80	7.41	1.81	7.36	1.81
112	13.73	6.42	13.31	6.53
128	16.96	18.88	17.27	19.01
192	40.77	240.55	39.75	239.76
256	81.52	1827.05	82.84	1834.13

Table 12: Execution time in milliseconds with ECIES and RSA-AES.

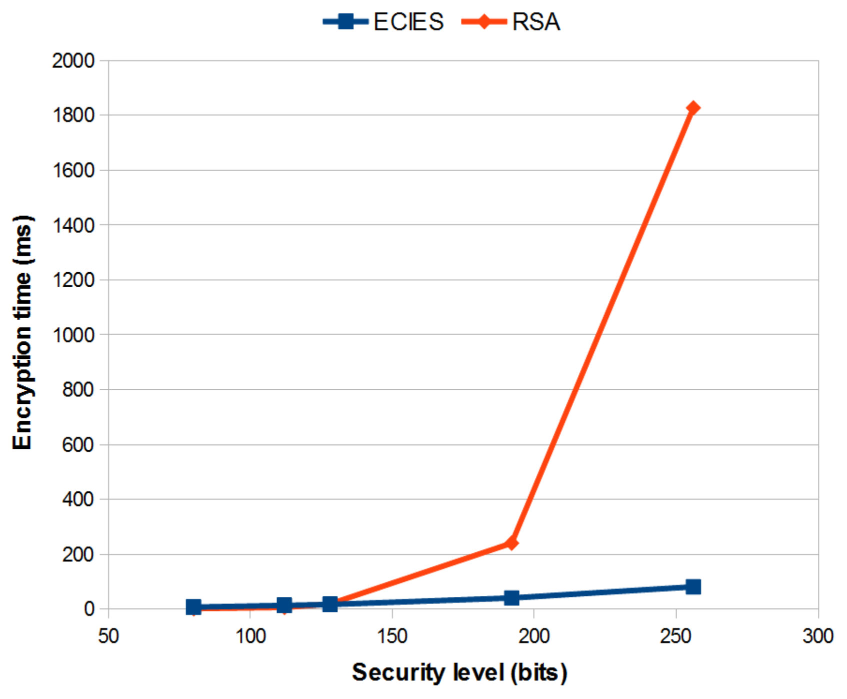


Figure 6: Encryption time comparison between ECIES and RSA-AES.

With all the previous data, it can be stated that ECIES is a valid alternative to RSA-AES as a hybrid encryption scheme, which combines the ease of key distribution associated to asymmetric cryptography and the good performance related to symmetric cryptography.

9 Conclusions

ECIES is the best known encryption scheme using elliptic curves, and as such it has been included in several standards. However, those standards offer a lot of options, which vary in the available functions and the specific settings of the scheme. This makes the selection of the proper configuration for a specific deployment scenario a difficult task.

Moreover, the number of options and the existence of internal dependencies in each standard provide as a consequence that, if the goal is to develop a practical and secure implementation of ECIES, there is no common set of functions and settings interoperable with the four standards analysed. We have shown with this contribution that, if a developer tries to implement the countermeasures for all the publicly known attacks on ECIES, the resulting version is interoperable only with two standards, IEEE 1363a and ISO/IEC 18033-2 or, alternatively, IEEE 1363a and SECG SEC 1, depending on one of the implementation decisions (see §6.2).

Taking into account all the security and efficiency considerations described along this contribution, we have selected a combination of algorithms and functionality options that create a secure implementation of ECIES. Then, we have tested the version of ECIES thus designed using a Java application. This application has allowed us to check the feasibility of our version of ECIES, as we were able to develop it using a popular programming language such as Java. The ECIES implementation has permitted us to compare the performance of the encryption process using different key lengths, which can be used to determine if the encryption scheme is adequate for key lengths related to high security levels. In order to provide a complete evaluation of ECIES, we have also developed in Java a comparable hybrid encryption scheme based on RSA and AES.

After conducting those tests it can be stated that, in absolute terms, the expansion added by the encryption process is not relevant when encrypting more than 1 KByte of information, which implies that ECIES is optimized (regarding bandwidth) for the encryption of medium to large messages. Encryption of small messages is also possible, but in that case the expansion produced by ECIES reduces its usefulness. When comparing the expansion factor of ECIES to that of our RSA-AES design, it is clear that ECIES generates smaller cryptograms.

With regards to the execution time of our Java implementation of ECIES, it shows almost a linear behaviour for small key lengths. As it was expected, elliptic curves with key lengths bigger than 256 bits present a slower performance, making the computation with those curves less efficient with today's technology. If we compare the results with the running time of our RSA-AES scheme, it can be affirmed that RSA-AES is faster for lower to medium security levels (i.e., smaller key lengths), but slower than ECIES for medium to higher security levels, which are the ones that will be used as time passes and new cryptographic attacks appear for both ECIES and RSA.

With all the gathered information we are in a position to conclude that, in its design as a hybrid encryption scheme, ECIES is a good alternative to RSA-AES for encrypting data that must be send through insecure communication networks.

Acknowledgment

This work has been partially supported by Ministerio de Ciencia e Innovación (Spain) under the grant TIN2011-22668, and by Comunidad de Madrid (Spain) under the project S2013/ICE-3095-CIBERDINE-CM.

References

[1] Abdalla, M., M. Bellare, and P. Rogaway (1998). *DHIES: An Encryption Scheme Based on the Diffie-Hellman Problem.* Contribution to IEEE P1363a. `http://cseweb.ucsd.edu/users/mihir/papers/dhaes.pdf`.

[2] Abdalla, M., M. Bellare, and P. Rogaway (2001). The Oracle Diffie-Hellman Assumptions and an Analysis of DHIES. *Lecture Notes in Computer Science 2020*, 143–158.

[3] Adams, C. (1997). *The CAST-128 Encryption Algorithm.* Internet Engineering Task Force, RFC 2144. `http://www.ietf.org/rfc/rfc2144.txt`.

[4] American National Standards Institute (2001). *Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography.* ANSI X9.63.

[5] Aoki, K., J. Guo, K. Matusiewicz, Y. Sasaki, and L. Wang (2009). Preimages for step-reduced SHA-2. *Lecture Notes in Computer Science 5912*, 578–597.

[6] Aoki, K., T. Ichikawa, M. Kanda, M. Matsui, S. Moriai, J. Nakajima, and T. Tokita (2001). Camellia: A 128-bit Block Cipher Suitable for Multiple Platforms - Design and Analysis. *Lecture Notes in Computer Science 2012*, 39–56.

[7] Bellare, M. and P. Rogaway (1997). Minimizing the Use of Random Oracles in Authenticated Encryption Schemes. *Lecture Notes in Computer Science 1334*, 1–16.

[8] Biryukov, A. and D. Khovratovich (2009). *Related-key Cryptanalysis of the Full AES-192 and AES-256*. Cryptology ePrint Archive, Report 2009/317. `http://eprint.iacr.org/2009/317.pdf`.

[9] Biryukov, A., M. Lamberger, F. Mendel, and I. Nikolic (2011). Second-order differential collisions for reduced SHA-256. *Lecture Notes in Computer Science 7073*, 270–287.

[10] Blake, I. F., G. Seroussi, and N. P. Smart (2004). *Advances in Elliptic Curve Cryptography*. Cambridge, UK: Cambridge University Press.

[11] Bogdanov, A., D. Khovratovich, and C. Rechberger (2011). *Biclique Cryptoanalysis of the Full AES*. Cryptology ePrint Archive, Report 2011/449. `http://eprint.iacr.org/2011/449.pdf`.

[12] Brainpool (2005). *ECC Brainpool Standard Curves and Curve Generation*. `http://www.ecc-brainpool.org/download/Domain-parameters.pdf`.

[13] Brainpool (2010). *Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation*. IETF RFC 5639. `http://tools.ietf.org/html/rfc5639`.

[14] Bundesamt für Sicherheit in der Informationstechnik (2009). *Elliptic Curve Cryptography*. BSI TR 03111. `http://www.bsi.de/literat/tr/tr03111/BSI-TR-03111.pdf`.

[15] Cannière, C. D., F. Mendel, and C. Rechberger (2007). Collisions for 70-step SHA-1: On the Full Cost of Collision Search. *Lecture Notes in Computer Science 4876*, 56–73.

[16] Dobbertin, H., A. Bosselaers, and B. Preneel (1996). RIPEMD-160: A Strengthened Version of RIPEMD. *Lecture Notes in Computer Science 1039*, 71–82.

[17] Dworkin, J. D., M. J. Torla, P. M. Glaser, A. Vadekar, R. J. Lambert, and S. A. Vanstone (2001). *Circuit and Method for Decompressing Compressed Elliptic Curve Points*. US Patent 6,199,086.

[18] Elo, T. (2000). Lessons Learned on Implementing ECDSA on a Java Smart Card. In *Proceedings of NordSec2000*. Reykjavik, Iceland.

[19] Gayoso Martínez, V., F. Hernández Álvarez, L. Hernández Encinas, and C. Sánchez Ávila (2010). A comparison of the standardized versions of ECIES. In *Sixth International Conference on Information Assurance and Security (IAS 2010)*, pp. 1–4.

[20] Gayoso Martínez, V., F. Hernández Álvarez, L. Hernández Encinas, and C. Sánchez Ávila (2011). Analysis of ECIES and other cryptosystems based on elliptic curves. *International Journal of Information Assurance and Security 6*(4), 285–293.

[21] Gayoso Martínez, V. and L. Hernández Encinas (2013). Implementing the ECC Brainpool curve generation procedure using open source software. In *WorldComp 2013 - International Conference on Security & Management - SAM'13*, pp. 162–197. Las Vegas, USA.

[22] Gayoso Martínez, V., L. Hernández Encinas, and C. Sánchez Ávila (2010). A Java Implementation of the Elliptic Curve Integrated Encryption Scheme. In *WorldComp 2010 - International Conference on Security & Management - SAM'10*, Volume II, pp. 495–501. Las Vegas, USA.

[23] Gueron, S., S. Johnson, and J. Walker (2010). *SHA-512/256*. Cryptology ePrint Archive, report 2010/548. `http://eprint.iacr.org/2010/548.pdf`.

[24] Hankerson, D., A. J. Menezes, and S. Vanstone (2004). *Guide to Elliptic Curve Cryptography*. New York, NY, USA: Springer-Verlag.

[25] Institute of Electrical and Electronics Engineers (2004). *Standard Specifications for Public Key Cryptography - Amendment 1: Additional Techniques*. IEEE 1363a.

[26] International Organization for Standardization / International Electrotechnical Commission (2004). *Information Technology – Security Techniques – Hash-functions – Part 3: Dedicated Hash-functions*. ISO/IEC 10118-3.

[27] International Organization for Standardization / International Electrotechnical Commission (2006). *Information Technology – Security Techniques – Encryption Algorithms – Part 2: Asymmetric Ciphers*. ISO/IEC 18033-2.

[28] Jia, Z. and Y. Zhang (2006). An Elliptic Curve Based User Authentication Scheme with Smart Cards. *Journal of Information Assurance and Security 1*, 283–292.

[29] Khovratovich, D., C. Rechberger, and A. Savelieva (2012). Bicliques for preimages: Attacks on Skein-512 and the SHA-2 family. *Lecture Notes in Computer Science 7549*, 244–263.

[30] Koblitz, N. (1987). Elliptic Curve Cryptosystems. *Mathematics of Computation 48*, 203–209.

[31] Krawczyk, H., M. Bellare, and R. Canetti (1997). *HMAC: Keyed Hashing for Message Authentication*. Internet Engineering Task Force, RFC 2104. `http://www.ietf.org/rfc/rfc2104.txt`.

[32] Lee, H. J., S. J. Lee, J. H. Yoon, D. H. Cheon, and J. I. Lee (2005). *The SEED Encryption Algorithm*. Internet Engineering Task Force, RFC 4269. `http://www.ietf.org/rfc/rfc4269.txt`.

[33] Lipmaa, H., P. Rogaway, and D. Wagner (2000). *Comments to NIST Concerning AES Modes of Operations: CTR-mode Encryption*. National Institute of Standards and Technology. `http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/ctr/ctr-spec.pdf`.

[34] Menezes, A. J. (1993). *Elliptic Curve Public Key Cryptosystems*. Boston, MA, USA: Kluwer Academic Publishers.

[35] Miller, V. S. (1986). Use of Elliptic Curves in Cryptography. *Lecture Notes in Computer Science 218*, 417–426.

[36] National Institute of Standards and Technology (2001). *Advanced Encryption Standard*. NIST FIPS 197.

[37] National Institute of Standards and Technology (2002). *The Keyed-hash Message Authentication Code*. NIST FIPS 198.

[38] National Institute of Standards and Technology (2005). *NIST Comments on Cryptanalytic Attacks on SHA-1*. National Institute of Standards and Technology. `http://csrc.nist.gov/groups/ST/hash/statement.html`.

[39] National Institute of Standards and Technology (2005a). *Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication*. NIST SP 800-38B.

[40] National Institute of Standards and Technology (2005b). *Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher*. NIST SP 800-67.

[41] National Institute of Standards and Technology (2007). *Recommendation for Key Management. Part 1: General*. NIST SP 800-57.

[42] National Institute of Standards and Technology (2012). *NIST Selects Winner of Secure Hash Algorithm (SHA-3) Competition*. National Institute of Standards and Technology. `http://www.nist.gov/itl/csd/sha-100212.cfm`.

[43] National Institute of Standards and Technology (2012). *Secure Hash Standard*. NIST FIPS 180-4.

[44] National Institute of Standards and Technology (2013). *Recommendation for Pair-wise Key Establishment Schemes Using Discrete Logarithm Cryptography*. NIST SP 800-56A rev. 2.

[45] Ohta, H. and M. Matsui (2000). *A Description of the MISTY1 Encryption Algorithm*. Internet Engineering Task Force, RFC 2994. `http://www.ietf.org/rfc/rfc2994.txt`.

[46] Oracle Corp. (2013a). *Java Platform SE 6 - System*. `http://docs.oracle.com/javase/6/docs/api/java/lang/System.html#nanoTime%28%29`.

[47] Oracle Corp. (2013b). *Java Technology*. `http://java.com/en/about`.

[48] Oracle Corp. (2013c). *Oracle Java Archive*. http://www.oracle.com/technetwork/java/archive-139210.html.

[49] Oracle Corporation (2012a). *Java Card Classic Platform Specification 3.0.4*. http://www.oracle.com/technetwork/java/javame/javacard/download/specs-jsp-136430.html.

[50] Oracle Corporation (2012b). *Java Card Platform Specification 2.2.2*. http://www.oracle.com/technetwork/java/javacard/specs-138637.html.

[51] Quisquater, J. and F. Koeune (2002). *ECIES - Security Evaluation of the Encryption Scheme and Primitives*. Cryptrec. http://www.ipa.go.jp/security/enc/CRYPTREC/fy15/doc/1015_ECIES_report.pdf.

[52] Rackoff, C. and D. R. Simon (1992). Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. *Lecture Notes in Computer Science 576*, 433–444.

[53] Rice, B. and B. Yankosky (2009). Elliptic Curve Cryptography with the TI-83. *Cryptologia 33*(2), 125–141.

[54] Sanadhya, S. and P. Sarkar (2008). New collision attacks against up to 24-step SHA-2. *Lecture Notes in Computer Science 5365*, 91–103.

[55] Schneier, B. (2011). *Schneier on Security*. http://www.schneier.com/blog/archives/2011/08/new_attack_on_a_1.html.

[56] Seroussi, G. (2001). *Compression and Decompression of Elliptic Curve Data Points*. US Patent 6,252,960.

[57] Shoup, V. (2001). *A Proposal for an ISO Standard for Public Key Encryption*. Cryptology ePrint Archive, Report 2001/112. http://www.shoup.net/papers/iso-2_1.pdf.

[58] Standards for Efficient Cryptography Group (1999). *Test vectors for SEC 1*. SECG GEC 2. http://www.secg.org/download/aid-390/gec2.pdf.

[59] Standards for Efficient Cryptography Group (2009). *Recommended Elliptic Curve Domain Parameters*. SECG SEC 1 ver. 2. `http://www.secg.org/download/aid-780/sec1-v2.pdf`.

[60] Stern, J. (2002). *Evaluation Report on the ECIES Cryptosystem*. Cryptrec. `http://www.ipa.go.jp/security/enc/CRYPTREC/fy15/doc/1016_Stern.ECIES.pdf`.

[61] Vanstone, S. A., R. C. Mullin, and G. B. Agnew (2000). *Elliptic Curve Encryption Systems*. US Patent 6,141,420.

[62] Wang, X., D. Feng, X. Lai, and H. Yu (2004). *Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD*. Cryptology ePrint Archive, report 2004/199. `http://eprint.iacr.org/2004/199.pdf`.

[63] Wang, X., X. Lai, D. Feng, H. Chen, and X. Yu (2005). Cryptanalysis of the Hash Functions MD4 and RIPEMD. *Lecture Notes in Computer Science 3494*, 1–18.

[64] Wang, X., Y. Yin, and H. Yu (2005). Finding Collisions in the Full SHA-1. *Lecture Notes in Computer Science 3621*, 17–36.