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

V. GAYOSO MARTÍNEZ, L. HERNÁNDEZ ENCINAS AND A. QUEIRUGA DIOS

Abstract

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, we provide the list of functions and options that must be used in such an implementation.

Keywords data encryption, elliptic curves, 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 [1], IEEE 1363a [2], ISO/IEC 18033-2 [3], and SECG SEC1 [4]. Those versions offer a high number of implementation options, making it impossible to identify a secure version of ECIES fully compatible with all the standards where ECIES is specified. 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. 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.

This paper is organized as follows: Sections 2 presents a brief introduction to Elliptic Curve Cryptography. 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 6 summarizes our proposed configuration for the encryption scheme. Finally, Section 7 provides our conclusions regarding the practical implementation of ECIES.

2 Elliptic Curve Cryptography

It is well known that Miller [5] and Koblitz [6] 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 $P = k \cdot G$ [7].

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 [8]. The ECDLP is the foundation of any cryptosystem based on elliptic curves, among which stands the Elliptic Curve Integrated Encryption Scheme (ECIES), included in several standards.

Due to its characteristics, ECC is particularly well-suited for devices with limited resources such as smart cards and some mobile devices [9–11].

The aim of this contribution is to present ECIES and identify the peculiarities of the different versions of that encryption scheme that have been standardised. With this 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 the results of an implementation of the proposed ECIES version using the Java programming language.

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 [12]:

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

where $a_1, a_2, a_3, a_4, a_6 \in \mathbb{F}_q$ and $\Delta \neq 0$, being $\Delta$ 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

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

- 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:
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 representing public keys.
- $n$ 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 [13], and it was later improved in [14] and [15], 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) [16].

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 operations or the key length [15].

DHIES represents the kernel of ECIES, which is the generic term used to define the best known encryption scheme using elliptic curves. It can be found in different flavours in the standards ANSI X9.63, IEEE 1636a, and ISO/IEC 18033-2, and in some deliverables from the Standards for Efficient Cryptography Group (SECG), e.g. SEC 1 and GEC 2 [17].

\[
y^2 + xy = x^3 + ax^2 + b.
\]
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.

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

In order to describe the steps that must be taken to encrypt a clear message, 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 $u$ 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, which is the product (optionally with the cofactor) of Alice’s ephemeral private key and Bob’s public key.

3. Take the shared secret value along with some optional parameters, identified as \( \text{Param} \ #1 \), 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 \( \text{Param} \ #2 \), 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 \( (U||\text{tag}||c) \), 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 two other 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:

---

**Figure 2** ECIES decryption process.
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 $u \cdot V = u \cdot v \cdot G = v \cdot u \cdot G = v \cdot U$ [7].

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 $\text{tag}^*$ 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 $\text{tag}^*$ 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 [18] 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. More specifically, given a cryptogram $(U||c||\text{tag})$, 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 $(U||c||\text{tag})$, the attacker is able to construct another valid cryptogram $(-U||c||\text{tag})$.

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 [18]. In this case, it is enough to add to the elliptic curve point $U$ an element whose order divides the cofactor $h$.

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.
One of the possible solutions was proposed by Shoup [18], and consists in adding the ephemeral public key $U$ to the input of the KDF function. Another recommendation consists in not using the DHC primitive in the generation of the shared secret, which implies that the DH primitive should be used instead. We provide our comments about these options in §5.1 and §6.2.

4.2 Malleability when using the XOR function

Shoup also proved in [18] 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 CCA2s. Some solutions to this problem are described below:

1. Establish a fixed length for all the plaintexts [18].
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 it is suggested in [14], [18], and [19].
3. Forbid the usage of stream ciphers in ECIES, allowing only block ciphers, as recommended in [18] and [20].

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.

Finally, if we add to the previous considerations the fact that using the XOR function with messages of variable length may represent a security risk in ECIES, 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 attacks is possible when an opponent deliberately provides an invalid public key [7]. 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 attacks are:

1. Check carefully the validity of the parameters and of the public key provided by the receiver (i.e., check that the order of the public key $V$ is $n$ [4]).
2. Use the DHC primitive instead of the DH function. 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 [18].
3. Replace the shared secret by the hash code of the secret value as the input to the KDF function [2].

4. Use the ephemeral public key of the sender with a KDF function that includes a hash primitive.

In a typical scenario, the validity check on the public key V would be performed by the trusted third party issuing certificates, so the 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, being that one of the reasons why most test vectors included in the standards do not use the DHC primitive.

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 [21]. However, this feature is not used in the test vectors of neither ISO/IEC 18033-2 nor SECG GEC 2, and Java Card 3.0 has added another operation mode in which the output of the KA function is not hashed.

The available information suggests that the small subgroup threat is cancelled by using the ephemeral public key of the sender (which avoids the KDF output to depend only on the values of the ephemeral private key u and the public key V) in conjunction with a KDF function that includes a hash primitive (conveniently masking the u · V value). This combined solution is not explicitly mentioned in the standards, although the usage of the ephemeral public key of the sender as input of the KDF is indeed included.

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 the different ECIES versions are:

- **Diffie-Hellman (DH):** This primitive consists in computing the product of the public key V and the private key u, thus obtaining the point of the curve \( P = (x_P, y_P) = u \cdot V \).

- **Diffie-Hellman with cofactor (DHC):** In this case, the cofactor of the curve h is used in the calculation, so the element obtained is \( P' = (x_{P'}, y_{P'}) = h \cdot u \cdot V \).
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 to use 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 list of hash functions mentioned in the standards where ECIES is included is:

- SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512 [22].
- RIPEMD-128 and RIPEMD-160 [23].
- WHIRLPOOL [24].

Table 1 presents the hash functions used in each standard.

<table>
<thead>
<tr>
<th>ANSI X9.63</th>
<th>IEEE 1363a</th>
<th>ISO/IEC 18033-2</th>
<th>SECG SEC 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA-1</td>
<td>SHA-1</td>
<td>SHA-1</td>
<td>SHA-1</td>
</tr>
<tr>
<td>SHA-224</td>
<td>SHA-256</td>
<td>SHA-256</td>
<td>SHA-224</td>
</tr>
<tr>
<td>SHA-256</td>
<td>SHA-384</td>
<td>SHA-384</td>
<td>SHA-256</td>
</tr>
<tr>
<td>SHA-384</td>
<td>SHA-512</td>
<td>SHA-512</td>
<td>SHA-384</td>
</tr>
<tr>
<td>SHA-512</td>
<td>RIPEMD-160</td>
<td>RIPEMD-128</td>
<td>SHA-512</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RIPEMD-160</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>WHIRLPOOL</td>
<td></td>
</tr>
</tbody>
</table>

Table 1 Hash functions.

In 2004, Xiaoyun Wang et al. created collision attacks against MD5, SHA-0, and other related hash functions [25, 26]. In February 2005, Wang et al. found a method to find collisions in the SHA-1 hash function, where the attack is estimated to require less than $2^{69}$ operations, fewer than the $2^{80}$ operations previously thought to be needed in order to find a collision in SHA-1 [27]. That year, the same authors announced at the CRYPTO conference rump session that they had reduced the complexity of the attacks from $2^{69}$ to $2^{63}$.

There exist other attacks against SHA-1, some of them related to boomerang attacks and coding theory. A survey of the different methods for fast collision search is presented in [28].

Due to these advances, 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 stronger SHA-2 family of hash functions, 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. After selecting five candidates for the final round of the competition (BLAKE, Gröstl, JH, Keccak, and Skein), on October 2012 NIST
finally announced Keccak as the new SHA-3 hash algorithm [29]. 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.

Regarding the security of the SHA-2 hash functions, even though they could potentially be attacked with techniques similar to the ones used against SHA-1, and although several reduced-step attacks have been proposed during the last years [30–32], so far no attack has been published against the full 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 [33].

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 to use SHA-256. In contrast, if memory and bandwidth are not critical elements, then we suggest to use SHA-512. Another argument in favour of selecting SHA-512 over SHA-256 is its better performance on 64-bit architectures [34], 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 [1].
- NIST-800-56-Concatenation-KDF [35].
- KDF1 and KDF2 [3].

The KDF functions considered in each standard version of ECIES are presented in Table 2:

<table>
<thead>
<tr>
<th>ANSI X9.63</th>
<th>IEEE 1363a</th>
<th>ISO/IEC 18033-2</th>
<th>SECG SEC 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>KDF2</td>
<td>NIST-800-56</td>
</tr>
</tbody>
</table>

**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 previous KDF functions, so theoretically any of them could be used in a secure implementation of ECIES. Given that the KDF2 algorithm (or, alternatively, 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 functions are 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, and HMAC-SHA-512 [36].
- HMAC-RIPEMD-128 and HMAC-RIPEMD-160 [37].
- CMAC-AES-128, CMAC-AES-192, and CMAC-AES-256 [38].

Tables 3 and 4 show the allowed MAC functions in the four standards.

<table>
<thead>
<tr>
<th>ANSI X9.63</th>
<th>IEEE 1363a</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMAC-SHA-1</td>
<td>HMAC-SHA-1</td>
</tr>
<tr>
<td>HMAC-SHA-224</td>
<td>HMAC-SHA-256</td>
</tr>
<tr>
<td>HMAC-SHA-256</td>
<td>HMAC-SHA-384</td>
</tr>
<tr>
<td>HMAC-SHA-384</td>
<td>HMAC-SHA-512</td>
</tr>
<tr>
<td>HMAC-SHA-512</td>
<td>HMAC-RIPEMD-160</td>
</tr>
</tbody>
</table>

Table 3 MAC functions (I).

<table>
<thead>
<tr>
<th>ISO/IEC 18033-2</th>
<th>SECG SEC 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMAC-SHA-1</td>
<td>HMAC-SHA-1-80/160</td>
</tr>
<tr>
<td>HMAC-SHA-256</td>
<td>HMAC-SHA-224-112/224</td>
</tr>
<tr>
<td>HMAC-SHA-384</td>
<td>HMAC-SHA-256-128/256</td>
</tr>
<tr>
<td>HMAC-SHA-512</td>
<td>HMAC-SHA-384-192/384</td>
</tr>
<tr>
<td>HMAC-RIPEMD-128</td>
<td>HMAC-SHA-512-256/512</td>
</tr>
<tr>
<td>HMAC-RIPEMD-160</td>
<td>CMAC-AES-128/192/256</td>
</tr>
<tr>
<td>HMAC-WHIRLPOOL</td>
<td></td>
</tr>
</tbody>
</table>

Table 4 MAC functions (II).

In case of using one of the SHA-2 functions as the hashing algorithm, we recommend to use one of MAC functions belonging to the HMAC-SHA-2 family. If the deployment scenario has memory and bandwidth limitations, we propose to use 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 list of symmetric algorithms included in the different versions of ECIES
is:

- XOR.
- Triple DES, also known as the Triple Data Encryption Algorithm (TDEA), in CBC (Cipher Block Chaining) mode [39].
- AES-128, AES-192, and AES-256, in CBC and CTR (Counter) modes [16].
- MISTY1 [40].
- CAST-128 [41].
- Camellia [42].
- SEED [43].

The symmetric ciphers considered in the standards that include ECIES are shown in Table 5.

<table>
<thead>
<tr>
<th>ANSI X9.63</th>
<th>IEEE 1363a</th>
<th>ISO/IEC 18033-2</th>
<th>SECG SEC 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>XOR</td>
<td>XOR</td>
<td>XOR</td>
<td>XOR</td>
</tr>
<tr>
<td>TDES/CBC/PKCS5</td>
<td>XOR</td>
<td>TDES/CBC</td>
<td>TDES/CBC</td>
</tr>
<tr>
<td>AES/CBC/PKCS5</td>
<td>AES/CBC/PKCS5</td>
<td>AES/CBC</td>
<td>AES/CBC</td>
</tr>
<tr>
<td>MISTY1/CBC/PKCS5</td>
<td>AES/CBC/PKCS5</td>
<td>AES/CTR</td>
<td></td>
</tr>
<tr>
<td>CAST-128/CBC/PKCS5</td>
<td>Camellia/CBC/PKCS5</td>
<td></td>
<td>SEED/CBC/PKCS5</td>
</tr>
<tr>
<td>Camellia/CBC/PKCS5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEED/CBC/PKCS5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 5** Symmetric encryption functions.

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 published as Rijndael. Each of these ciphers has a 128-bit block size, with key sizes of 128, 192, and 256 bits, respectively.

On December 2009, Biryukov and Khovratovich published an improvement of their initial key recovery attack on the related key setting against the full versions of AES-192 and AES-256 that works for all the keys and has a time and data complexity of $2^{99.5}$ [44]. As the authors recognize [45], their attack cannot be applied to AES-128.

In 2011, Bogdanov et al. created a new single-key attack on the full AES cipher which is faster than brute force [46]. This attack is based on the so-called biclique cryptanalysis, and requires $2^{126.1}$ operations to recover an AES-128 key. For AES-192 and AES-256, $2^{189.7}$ and $2^{254.4}$ operations are needed, respectively. Though this new attack represents an important advance, the authors state that “as our attacks are of high computational complexity, they do not threaten the practical use of AES in any way” [46].
As a summary of the information presented in the previous paragraphs, it is commonly agreed that, until new attacks are published, AES-128 is relatively more secure than AES-192 and AES-256 [47].

Regarding the mode of operation, even though CTR offers some advantages (it does not require padding, its implementation is more efficient, etc.) [48], as it is only considered as 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.

5.6 Summary of functions allowed in all the standards

As a summary of the information that has been previously presented, Table 6 includes all the cryptographic functions and algorithms allowed simultaneously in the four standard versions of ECIES cited along this document.

<table>
<thead>
<tr>
<th>KA</th>
<th>HASH</th>
<th>KDF</th>
<th>ENC</th>
<th>MAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>DH</td>
<td>SHA-1</td>
<td>KDF2</td>
<td>XOR</td>
<td>HMAC-SHA-1</td>
</tr>
<tr>
<td>DHC</td>
<td>SHA-256</td>
<td></td>
<td></td>
<td>HMAC-SHA-256</td>
</tr>
<tr>
<td></td>
<td>SHA-384</td>
<td></td>
<td></td>
<td>HMAC-SHA-384</td>
</tr>
<tr>
<td></td>
<td>SHA-512</td>
<td></td>
<td></td>
<td>HMAC-SHA-512</td>
</tr>
</tbody>
</table>

Table 6 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 for devices with limited resources is to use the following set of functions in order to obtain a secure version of ECIES: DH, SHA-256, KDF2, AES-128, and HMAC-SHA-256. If memory and processing capabilities are not an issue in the deployment scenario, then we propose to use the following set of functions: DH, SHA-512, KDF2, AES-128, and HMAC-SHA-512.

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 by 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 are the following:

1. **Uncompressed**: Both coordinates are taken into account. A header byte 0x04 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 0x02 or 0x03. 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 [19, 20]. However, even though Miller suggested compressing a public key to simply its first coordinate [5], there are several patents over this topic [49–51], so in order not to infringe any of those patents it would be recommended 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, so they can use the following elements:

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

2. The element selected given the previous decision (i.e., \( P \), \( P' \), \( x_P \) or \( x_{P'} \)), or the hash output of that element.

3. The point that represents the ephemeral public key, either compressed or uncompressed, together with the previous data.

4. Additional parameters.

Table 7 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 \( U \) (the sender’s ephemeral public key), 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 security implications of using the whole point representing the shared secret, or only the first coordinate, some authors, such as Stern [19], state that from 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.

As for the decision of using the shared secret or its hashed value, its benefits and disadvantages have been already presented in §4.3, so in this case we suggest to use the shared secret $x$-coordinate instead of its hash value.

Finally, regarding 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 ($k_{ENC}||k_{MAC}$).

<table>
<thead>
<tr>
<th>ANSI X9.63</th>
<th>IEEE 1363a</th>
<th>ISO/IEC 18033-2</th>
<th>SECG SEC 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_P$</td>
<td>$x_P$</td>
<td>$x_P$</td>
<td>$x_P$</td>
</tr>
<tr>
<td>$x_P, P#1$</td>
<td>$x_P, P#1$</td>
<td>$U</td>
<td></td>
</tr>
<tr>
<td>$U</td>
<td></td>
<td>x_P$</td>
<td>$x_U</td>
</tr>
<tr>
<td>$x_U</td>
<td></td>
<td>x_P$</td>
<td></td>
</tr>
<tr>
<td>$x_U</td>
<td></td>
<td>x_P, P#1$</td>
<td></td>
</tr>
</tbody>
</table>

Table 7 KDF input data options.
All the analysed standards allow the $k_{ENC}||k_{MAC}$ interpretation. Only IEEE 1363a permits to use the $k_{MAC}||k_{ENC}$ interpretation, and strictly under specific circumstances (stream cipher, etc.).

In any case, $k_{ENC}||k_{MAC}$ is the interpretation used by all the standards when using a block cipher as the symmetric encryption algorithm, and as we have already mentioned the advantages of this kind of symmetric functions in §4.2, $k_{ENC}||k_{MAC}$ is the option that must be chosen for any practical ECIES implementation.

6.4 MAC input data

The four standards allow to use 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 strengthened if sender and receiver share the content of this parameter (e.g. a passphrase), so using it is therefore recommended. 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, such as Shoup, consider that it is 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 [18], and in some standards as 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 [20].

However, as the requirement made by Shoup does not seem to have negative implications, we adhere to the recommendation of not 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 element 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 [52]:

- 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” [52], 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” [53].

The Brainpool specification include 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 to use 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, and it does not affect the interoperability of the ECIES implementation.

7 ECIES configuration

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

- KA: The DH function.
- Hash: SHA-512 (SHA-256 in devices with limited resources).
- KDF: KDF2.
- ENC: AES-128 in CBC mode.
- MAC: HMAC-SHA-512 (HMAC-SHA-256 in devices with limited resources).
- Shared secret: Use only the first coordinate (without hash).
- Input to the KDF function: Include the 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).
- Elliptic curve generation procedure: Brainpool.
8 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, both related to the available functions and the specific settings of the scheme, which 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. We have shown along 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 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.

Acknowledgments

This work has been partially supported by Ministerio de Ciencia e Innovación (Spain) under the grant TIN2011-22668, and by Fundación Memoria D. Samuel Solórzano Barruso under the project FS/19-2011.

References


21
