ECIES-KEM vs. PSEC-KEM

Alexander W. Dent*

NES/DOC/RHU/WP5/028/2

1 Introduction

The purpose of this paper is to discuss the similarities and differences between the PSEC-KEM and ECIES-KEM. The schemes are in very similar in some ways: both base their security on the Diffie-Hellman key-agreement protocol and both make heavy use of the random oracle model. However there are a few very important differences: PSEC-KEM is an authenticated KEM whilst ECIES-KEM is unauthenticated, and ECIES-KEM use the Diffie-Hellman key-agreement protocol directly to compute the key whereas PSEC-KEM uses the Diffie-Hellman protocol to compute a mask for a randomly generated key.

These differences lead to a major difference in their security proofs: ECIES-KEM reduces to the gap Diffie-Hellman problem [5] whilst PSEC-KEM reduces to the weaker computational Diffie-Hellman problem.

We will assume that the reader is familiar with the concepts of KEM-DEM constructions and their security proofs. For more information the reader is referred to [3, 4]. Briefly the security of a KEM is defined by the advantage an attacker has in winning a game played against a mythical system. The game is played as follows:

1. The system generates a public and secret key
$$(pk, sk) = KEM.KeyGen(\lambda).$$
2. The attacker runs until it is ready to receive a challenge encapsulation. In this time it is allowed to query a decapsulation oracle for the KEM.
3. The system generates a challenge ciphertext as follows:
   (a) The system generates a valid encapsulation
   $$(K_0, \psi) = KEM.Encapsulate(r, pk)$$
   for some randomly generated r.
   (b) The system randomly generates a key K_1.
   (c) The system randomly chooses a bit $\sigma \in \{0, 1\}$.

 This paper has been supported by the Commission of the European Communities through the IST programme under contract IST-1999-12324.

(d) The system returns (K_σ, ψ) to the attacker.

4. The attacker outputs a guess σ' for σ. Whilst running the attacker is allowed to query a decapsulation oracle for the KEM, with the exception that it is not allowed to ask for the decapsulation of the challenge encapsulation ψ.

The attacker wins the game if $\sigma' = \sigma$ and the attackers advantage is defined to be $Pr[\sigma' = \sigma] - 1/2$. The advantage of the KEM is the maximum advantage of any polynomial-time attacker. This is called the IND-CCA2 security of the algorithm.

2 ECIES-KEM

First we define the ECIES-KEM scheme. Note that we implement the probabilistic encryption algorithm as a deterministic algorithm that takes as input a random seed r that satisfies $1 \leq r \leq p-1$.

Key Generation. ECIES-KEM, like ECIES, is an elliptic curve scheme. This means that before the scheme can be implemented a suitably secure elliptic curve E must have been generated and a point $P \in E$ with prime order p must have been chosen. We assume that the length of p is equal to the security parameter λ. The key generation algorithm is a probabilistic algorithm that takes the elliptic curve parameters (E, P, p, λ) as input.

1. Generate (usually randomly) an integer $s \in \{1, \ldots, p-1\}$.
2. Set $W := sP$.
3. Set $pk := (E, P, p, W, \lambda)$ and $sk := (s, pk)$.
4. Output the key-pair (pk, sk).

Encapsulation Algorithm. The encapsulation algorithm is a probabilistic algorithm that takes the public-key as input. It uses a public and pre-agreed key derivation function $KDF(\cdot)$ that must be available to all parties wishing to use the scheme. Its encapsulation algorithm runs as follows.

1. Generate a random integer $r \in \{1, \ldots, p-1\}$.
2. Set $C := rP$.
3. Set x to be x-coordinate of rW.
4. Set $K := KDF(C||x)$.
5. Output the encapsulated key-pair (K, C).

Decapsulation Algorithm. The decapsulation algorithm is a deterministic algorithm that takes as input an encapsulated key C and the secret-key sk. It also uses the pre-agreed key derivation function $KDF(\cdot)$ that was used in the encapsulation process. The algorithm runs as follows.

1. Set $Q := sC$.
2. Check that $Q \neq \mathcal{O}$. If so, output `Invalid Ciphertext` and halt.

3. Set x to be the x-coordinate of Q.
4. Set $K := KDF(C||x)$.
5. Output K.

It is pretty easy to show that ECIES-KEM is IND-CCA2 secure in the random oracle model [1] providing the gap Diffie-Hellman problem [5] is hard. The random oracle model involves modelling certain parts of a scheme, in this case the key-derivation function, as purely random functions. This is considered to be a good heuristic argument as to the security of the scheme but it also been shown that this does not constitute a formal proof of security [2].

Theorem 1. *Suppose there exists an attacker $\mathcal{A}$ against the IND-CCA2 security of ECIES-KEM that has advantage ϵ, runs in time at most t and makes at most q_D queries to the decryption oracle and q_K queries to the random oracle that models the key-derivation function. Then there exists an algorithm $\mathcal{B}$ that runs in time at most t' and solves the gap Diffie-Hellman problem over elliptic curves with probability ϵ' with*

$$\epsilon' = \epsilon \tag{1}$$

$$t' \approx t + 2q_K T \ . \tag{2}$$

where T is the time taken for the gap oracle to check the validity of a Diffie-Hellman quadruple.

Proof. In order to solve a gap Diffie-Hellman problem we simulate an instance of the IND-CCA2 game and execute $\mathcal{A}$. Suppose we are given a triple (P, aP, bP) of points on the elliptic curve E where P is of prime order p, and that we wish to compute abP.

Firstly we prepare the simulation of the IND-CCA2 game by setting $W := aP$ and $pk := (E, P, p, W, \lambda)$ where λ is equal to the size of p. Next we prepare two lists: $DecList$ and $KDFList$ and set them both to be empty. Now we allow $\mathcal{A}$ to run until it is ready to receive a challenge.

If, during this time, $\mathcal{A}$ asks for the decryption query for an encapsulation ψ then we do one of two things things. If (ψ, K) exists on the list $DecList$ for some K then we return K. If not, then randomly select some appropriately sized K, add (ψ, K) to $DecList$ and return K.

However if $\mathcal{A}$ queries the random oracle representing the key derivation function with an input X then we must be a little more subtle to maintain consistency:

1. If (X, K) exists on the list $KDFList$ then we return K.
2. If not then we attempt to parse X as $\psi||x$ where ψ is an elliptic curve point and x is the x-coordinate of an elliptic curve point. If we cannot do this then we randomly select some appropriately sized K, add (X, K) to $KDFList$ and return K.
3. Next we find points Q and $-Q$ that have an x-coordinate equal to x and check to see if $(P, aP, \psi, \pm Q)$ is a valid Diffie-Hellman triple using the gap

oracle (i.e. we check to see if there exists c such that $\psi = cP$ and $Q = acP$). If not then we randomly select some appropriately sized K, add (X, K) to $KDFList$ and return K.

4. Let us assume that (P, aP, ψ, Q) is a valid Diffie-Hellman triple. If $\psi = bP$ then we terminate completely, returning Q as the answer to the gap Diffie-Hellman problem.
5. Lastly we conclude that if $\mathcal{A}$ were to ask for the decryption of ψ then it should be given $KDF(X)$ so we randomly select some appropriately sized K, add (X, K) to $KDFList$, (ψ, K) to $DecList$ and return K.

When $\mathcal{A}$ is ready to receive a challenge encapsulation we randomly select an appropriately sized K and return (K, ψ). We then allow $\mathcal{A}$ to run to completion, answering any queries it makes as before (with the obvious exception that it cannot ask for the decryption of bP).

Let E be the event that the random oracle is queried with $bP||x$ where x is the x-coordinate of abP. Until E occurs, $\mathcal{A}$ has no advantage in working out whether (K, bP) is a valid encapsulation pair as it does not know the value of the output of the key derivation function. The simulation is completely consistent up until this point. When the event E does occur then we terminate the algorithm completely and return the value of abP. The probability that E occurs is at least ϵ as $\mathcal{A}$ has advantage ϵ so the probability that $\mathcal{B}$ succeeds is ϵ.

We note that the running time of $\mathcal{B}$ is approximately equal to the running time of $\mathcal{A}$ plus the time taken to check all the Diffie-Hellman quadruples. □

Note that a similar proof appears in [3].

3 Re-writing ECIES-KEM

As it stands ECIES-KEM is optimised for implementation, hence the use of the x-coordinate x in the evaluation of the key-derivation function rather than the representation of the whole point Q. We find that it is convenient, when comparing ECIES-KEM to PSEC-KEM, to use the variant of ECIES-KEM in which the whole point Q is used as the input to the key-derivation function. In this section we will show that these two forms have equivalent security. We do this using a "game hopping" technique.

Game 1 This is the scheme as specified in the previous chapter, in the random oracle IND-CCA2 attack model.

Game 2 Here we alter the encryption/decryption procedure so that the key-derivation function $KDF()$ takes the entire point as input, not just the x-coordinate. Formally the encryption algorithm is now:

1. Set $C := rP$.
2. Set $Q := rW$.
3. Set $K := KDF(C||Q)$.
4. Output the encapsulated key-pair (K, C).

and the decryption algorithm is of the form

1. Set $Q := sC$.
2. Check that $Q \neq \mathcal{O}$. If so, output `Invalid Ciphertext` and halt.
3. Set $K := KDF(C||Q)$.
4. Output K.

We still attack this scheme in the random oracle IND-CCA2 model.

Lemma 1. *If there exists an attacker $\mathcal{A}_1$ that breaks the scheme given in Game 1 with advantage Adv then there exists an attacker $\mathcal{A}_2$ that breaks the scheme given in Game 2 with advantage Adv.*

Proof. We use the normal trick of constructing an algorithm to break the scheme given in Game 2 that uses the algorithm $\mathcal{A}_1$ as a subroutine. It is easy to see that the whenever $\mathcal{A}_1$ asks for a decryption or for the challenge then we can just request the answers from the system. The only change occurs when $\mathcal{A}_1$ requests the value $KDF(X)$ from the random oracle.

If the request is not of the form $C||x$ or x is not a valid x-coordinate for an elliptic curve point then we can just pass back a randomly generated value, keeping a list of pairs $(X, KDF(X))$ for consistency.

Let the decapsulation oracle be $\mathcal{D}$.

Suppose that $\mathcal{A}_1$ queries the key-derivation function on an input $X = C||x$ where x is the x-coordinate of an elliptic curve point. Now we can compute points T_1 and T_2 that have an x-coordinate x. We can then request $KDF(C||T_1)$ and $KDF(C||T_2)$, and must now decide which of these to pass back to $\mathcal{A}_1$. In order to do this we request the decapsulation of C, noting that if $C||T_i$ is a 'proper' input to the key-derivation function then $KDF(C||T_i) = \mathcal{D}(C)$ for some $i \in \{1, 2\}$. If this is the case then we pass this value back to $\mathcal{A}_1$ otherwise we keep a record of the query as before. Note that we will never have a situation where $C||T_1$ and $C||T_2$ are both 'proper' inputs to the key-derivation function as for each there exists only one point T such that $T = sC$.

Hence we have successfully simulated the Game 1 environment and so $\mathcal{A}_1$ will solve the IND problem correctly with probability Adv. Hence our algorithm correctly solves the IND problem with probability Adv.

We have expanded the number of oracle queries made though. If $\mathcal{A}_1$ makes q_D decapsulation queries and q_K key-derivation function queries then $\mathcal{A}_2$ makes at most $q_D + q_K$ decapsulation queries and $2q_K$ key-derivation function queries. □

It is trivial to show that an attacker in Game 2 with advantage Adv implies the existence of an attacker in Game 1 with advantage Adv just by stripping all queries to the key-derivation function of their x-coordinates.

Game 3 We alter the algorithm used in Game 2 by removing the exclusion of the point $\mathcal{O}$ in the decryption algorithm. This exclusion was put in place to avoid attempting to compute the x-coordinate of $\mathcal{O}$, which doesn't exist. Formally, the decryption algorithm is now defined as

1. Set $Q := sC$.

2. Set $K := KDF(C||Q)$.
3. Output K.

The attacker is still running in the IND-CCA2 attack model.

Lemma 2. *If there exists an attacker $\mathcal{A}_2$ that breaks the scheme defined in Game 2 with advantage Adv then there exists an attacker $\mathcal{A}_3$ that breaks the scheme defined in Game 3 with advantage Adv.*

Proof. Again we construct an algorithm $\mathcal{A}_3$ that uses $\mathcal{A}_2$ as a subroutine. We can pass any queries that $\mathcal{A}_2$ makes direct to the system and pass the answers back to $\mathcal{A}_2$ unless it queries the decapsulation oracle on a ciphertext C such that $sC = \mathcal{O}$. Now this will only happen if $C = \mathcal{O}$ as $s < p$ and the group has prime order. Hence we just return $\perp$ if we are queried with a ciphertext $C = \mathcal{O}$ and return normal decryptions otherwise. □

Similarly if there exists an attacker in Game 3 then we can construct an attacker in Game 2 just by generating our own random 'ciphertext' to be returned whenever the attacker queries the decapsulation oracle on a ciphertext $C = \mathcal{O}$.

Theorem 2. *The security of the scheme defined in Game 1 (in the random oracle IND-CCA2 attack model) is the same as the security of the scheme defined in Game 3.*

4 PSEC-KEM

As has already been mentioned, PSEC-KEM is very similar to ECIES-KEM but PSEC-KEM is authenticated KEM (i.e. it rejects improperly formed ciphertexts) and ECIES-KEM is an unauthenticated KEM (i.e. it decrypts all ciphertexts). We start by giving the specifications for PSEC-KEM below. Again we write the encapsulation algorithm as a deterministic algorithm that takes some fixed length random seed as input.

Key Generation Since PSEC-KEM is defined over an elliptic curve, a suitable curve E will have to be generated before the key generation algorithm is executed. The curve should have a point P that generates a secure cyclic subgroup of E with prime order p. We assume that the p is of size λ where λ is the security parameter.

The key generation algorithm is a probabilistic algorithm that takes the elliptic curve E, the point P and the order p of P as input.

1. Generate an integer s uniformly at random from $\{0, \ldots, p-1\}$.
2. Set $W := sP$.
3. Set $pk := (E, P, W, p, \lambda)$ and $sk := (s, pk)$.
4. Output the key-pair (pk, sk).

Encapsulation Algorithm The key encapsulation algorithm is a written as a deterministic algorithm that takes as input a fixed length random seed r and the public-key pk. In order for the scheme to work the two communicating parties must have agreed the use of a common, public key derivation function $KDF(\cdot)$.

1. Set $H := KDF(0_{32}||r)$, where 0_{32} is the 32-bit representation of the integer 0.
2. Parse H as $t||K$ where t is an $(\lambda + 128)$-bit integer and K is a suitably sized symmetric key.
3. Set $\alpha := t \bmod p$.
4. Set $Q := \alpha W$.
5. Set $C_1 := \alpha P$.
6. Set $C_2 := r \oplus KDF(1_{32}||C_1||Q)$, where 1_{32} is the 32-bit representation of the integer 1.
7. Set $C := (C_1, C_2)$.
8. Output the encapsulated key-pair (K, C).

Decapsulation Algorithm The decapsulation algorithm is a deterministic algorithm that takes as input a key encapsulation C and the secret-key sk. It also uses the pre-agreed key derivation function $KDF(\cdot)$.

1. Parse C as (C_1, C_2).
2. Set $Q := sC_1$.
3. Set $r := C_2 \oplus KDF(1_{32}||C_1||Q)$, where 1_{32} is the 32-bit representation of the integer 1.
4. Set $H := KDF(0_{32}||r)$, where 0_{32} is the 32-bit representation of the integer 0.
5. Parse H as $t||K$ where t is an $(\lambda + 128)$-bit integer and K is a suitably sized symmetric key.
6. Set $\alpha := t \bmod p$.
7. Check $C_1 = \alpha P$. If not, output `Invalid Ciphertext` and halt.
8. Output K.

We will show that PSEC-KEM can be viewed as an algorithm whose security is very much based on an instance of ECIES-KEM. We start by examining the key derivation functions.

As a notational device we set

$$KDF_i(X) = KDF(i_{32}||X)$$

where i_{32} is the 32-bit representation of the integer i. As we model the key derivation functions as random oracles we can conclude that for any $i \neq j$ the outputs of KDF_i and KDF_j are completely uncorrelated. Hence PSEC-KEM actually uses two separate independent random oracles.

Next we note that PSEC-KEM uses a Diffie-Hellman key agreement protocol in a very similar way to ECIES-KEM. If we consider the version of ECIES-KEM that we defined in the last section then we may write PSEC-KEM as a scheme that uses an instantiation of ECIES-KEM as a subroutine. The encapsulation algorithm can be written as

1. Set $H := KDF_0(r)$.
2. Parse H as $t||K$ where t is an $(\lambda + 128)$-bit integer and K is a suitably sized symmetric key.
3. Set $\alpha := t \bmod p$.
4. Set $(X, C_1) := ECIES.Encapsulate(\alpha, pk)$.
5. Set $C_2 := r \oplus X$.
6. Set $C := (C_1, C_2)$.
7. Output the encapsulated key-pair (K, C).

and the decapsulation algorithm can be written as

1. Parse C as (C_1, C_2).
2. Set $X := ECIES.Decapsulate(C_1, sk)$.
3. Set $r := C_2 \oplus X$.
4. Set $H := KDF_0(r)$.
5. Parse H as $t||K$ where t is an $(\lambda + 128)$-bit integer and K is a suitably sized symmetric key.
6. Set $\alpha := t \bmod p$.
7. Check $C_1 = \alpha P$. If not, output `Invalid Ciphertext` and halt.
8. Output K.

The security proof for PSEC-KEM can be found in [6]. It shows that, in the random oracle model, the problem of breaking the system can be reduced to the problem of solving a computational Diffe-Hellman problem on the elliptic curve E. Formally, if there exists a (t, ϵ, q_D) IND-CCA2 attacker for PSEC-KEM then there exists a (t', ϵ') solver for the CDH problem on E with

$$\epsilon' \approx \frac{1}{q_D + q_{K_1}} \left\{ \epsilon - \frac{q_{K_0} + 2q_D}{p} - \frac{q_{K_0} + q_D}{2^\lambda} \right\} \tag{3}$$

$$t' \approx t \tag{4}$$

where

- the encryption algorithm generates a random integer of length λ in step 1,
- the attacker makes at most q_{K_i} queries to the random oracle representing the key-derivation function where the initial 32-bits are a representation of the integer i.

5 Reducing the security of PSEC-KEM to ECIES-KEM

The security of ACE-KEM, which is also based on the difficulty of solving Diffie-Hellman problems in elliptic curve groups, can be directly shown to be at least as secure as ECIES-KEM. In other words an algorithm that breaks ACE-KEM can be adapted to break ECIES-KEM with the same advantage. It would be nice if this were true for PSEC-KEM but this seems unlikely. This is because in order to change the ECIES-KEM challenge into a PSEC-KEM challenge we would have to calculate a suitable K and C_2 from the ECIES challenge (X, C_1)

but this does not seem possible without knowing a random seed r that is mapped to a string $t||K$ under the action of KDF_0 where t is the (unknown) random seed used by the ECIES-KEM algorithm.

We can suggest that the security of PSEC-KEM in a stronger model can be reduced to that of ECIES-KEM. Firstly we need to argue that the first use of the key derivation function in PSEC-KEM is unnecessary. The use of KDF_0 in PSEC-KEM only seems to serve one purpose - it expands the size of the fixed length random seed r to a longer random string $t||K$. So, instead of taking in a small string and expanding it, we take a longer string and just split it straight into $t||K$. This does not affect the distribution of the possible strings $t||K$ so it has no affect on the security of the scheme (although it does slightly improve the security bound). In this version of the scheme the ciphertext C_2 is formed by XORing the output from the ECIES-KEM challenge X with $t||K$. The problem that we do not the value of t would not be a problem if we only allow the attacker to have access to the last $|K|$-bits of C_2. So if there exists an algorithm that breaks PSEC-KEM when given the challenge encapsulated key-pair $(K, (C_1, C_2'))$ (where C_2' is the last $|K|$-bits of C_2) then there exists an algorithm that breaks ECIES-KEM.

So we can then prove that if there exists an algorithm that breaks PSEC-KEM given the last $|K|$-bits of C_2 then there exists an algorithm that breaks PSEC-KEM in the normal sense and there exists an algorithm that breaks ECIES-KEM in the normal sense. However this only means that if ECIES-KEM is secure then there can exist no algorithm that breaks PSEC-KEM and uses only the last $|K|$-bits of C_2, not that there exists no algorithm that breaks PSEC-KEM.

Of course, the we can indirectly show that if ECIES-KEM is secure then PSEC-KEM is secure by noting that if there exists an algorithm that breaks PSEC-KEM in a group then there exists an algorithm that solves the CDH problem in that group. Obviously an algorithm that solves the CDH problem in a group can be adapted to break to break ECIES-KEM. However this indirect reduction is not as efficient as we might have hoped from a direct reduction.

6 The randomness of the key-derivation function

Both schemes rely heavily on the random nature of the key-derivation function for their security. Let us consider what happens if there exists some kind of bias in the outputs of the key-derivation function. In other words we assume that, for random inputs, some patterns of output of the KDF occur with a frequency that is greater than would be expected for a random function. For this section we will consider the scheme as public-key encryption schemes - i.e. implemented with some suitable DEM.

Any bias in the output of the key-derivation function used in ECIES-KEM will directly correspond to a bias in the key used by the DEM.

PSEC-KEM however uses two key-derivation functions: KDF_0 and KDF_1. Any bias in KDF_0 will correspond to either a direct bias of the key or a bias in the

input to the ECIES-KEM component. Such a bias may or may not cause there to exist a bias in the output of ECIES-KEM component of the algorithm. A bias in KDF_1 will obviously cause there to be a bias in the output of the ECIES-KEM component of the algorithm. Such a bias may reveal some information about r and hence may lead to some information being discovered about the key K used by the DEM.

Let us consider the simpler version of PSEC-KEM in which KDF_0 has been removed. Here a bias in the output of the ECIES-KEM component of the algorithm will only directly reveal information about the key if the bias occurs in the first $|K|$-bits of the output. Of course bias in the remaining bits may reveal information about the random seed α used by the ECIES-KEM component, which may in turn reveal a bias in the first $|K|$-bits of the output or some further bias in the remaining bits and so on.

However it would seem that PSEC-KEM relies less heavily on the randomness of the key-derivation function than ECIES-KEM.

7 Conclusion

Whilst ECIES-KEM and PSEC-KEM are not directly comparable in security terms, it is clear that PSEC-KEM relies on weaker security assumptions than ECIES-KEM and so is a more secure algorithm. Of course the price for this increased security is a slower decryption algorithm, as PSEC-KEM has to compute one more elliptic curve multiplication than ECIES-KEM.

References

1. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proc. of the First ACM Conference on Computer and Communications Security*, pages 62–73, 1993.
2. R. Canetti, O. Goldreich, and S. Halvei. The random oracle model, revisited. In *Proceedings of the 30th Annual ACM Symposium on the Theory of Computing*, pages 209–218, 1998.
3. R. Cramer and V. Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. Technical report, http://shoup.net/, 2002.
4. A W. Dent. ACE-KEM and the general KEM-DEM construction. NESSIE technical report, `NES/DOC/RHU/WP5/028`, 2002.
5. T. Okamoto and D. Pointcheval. The gap problems: A new class of problems for the security of cryptographic schemes. In *Public Key Cryptography*, LNCS 1992, pages 104–118. Springer-Verlag, 2001.
6. V. Shoup. A proposal for the ISO standard for public-key encryption (version 2.0). Available from `http://shoup.net/`, 2001.