A Designer's Guide to KEMs

Alexander W. Dent

Information Security Group,
Royal Holloway, University of London,
Egham Hill, Egham, Surrey, U.K.
alex@fermat.ma.rhul.ac.uk
http://www.isg.rhul.ac.uk/~alex/

Abstract. A generic or KEM-DEM hybrid construction is a formal method for combining asymmetric and symmetric encryption techniques to give an efficient, provably secure public-key encryption scheme. This method combines an asymmetric KEM with a symmetric DEM, and each of these components must satisfy its own security conditions. In this paper we describe generic constructions for provably secure KEMs based on lower level primitives such as one-way trapdoor functions and simple key-agreement protocols.

1 Introduction

Whilst most dedicated public-key encryption algorithms are fine for sending short messages, many schemes have problems sending long or arbitrary length messages. Most of the normal "modes of operation" which might allow a sender to send a long message using a public-key encryption algorithm directly are cripplingly inefficient.

One particular way to solve these problems is to use symmetric encryption with a randomly generated key to encrypt a message, and then use asymmetric cryptography to encrypt that (short) random key. This method has been cryptographic folklore for years and, as such, was not formally studied. This led to papers such as [2] which can be used to attack schemes in which the set of symmetric keys is significantly smaller than the message space of the asymmetric scheme used to encrypt them. This folklore has recently been formalised in terms of a generic or KEM-DEM construction [3]. In this construction the encryption scheme is divided into two parts: an asymmetric KEM and a symmetric DEM. A KEM (or key encapsulation mechanism) is a probabilistic algorithm that produces a random symmetric key and an encryption of that key. A DEM (or data encapsulation mechanism) is a deterministic algorithm that encrypts a message of arbitrary length under the key given by the KEM.

In this paper we provide generic methods for constructing KEMs from low level components such as encryption functions and key-agreement protocols. Furthermore we provide security proofs that show that the KEM will be highly secure provided these lower level components have certain weak security properties. Essentially this paper gives a toolbox to allow an algorithm designer to construct a KEM from almost any cryptographic problem.

2 Preliminaries

A KEM is a triple of algorithms:

– a key generation algorithm, $KEM.Gen$, which takes as input a security parameter 1^λ and outputs a public/secret key-pair (pk, sk),
– a encapsulation algorithm, $KEM.Encap$, that takes as input a public-key pk and outputs an encapsulated key-pair (K, C) (C is sometimes said to be an encapsulation of the key K),
– a decapsulation algorithm, $KEM.Decap$, that takes as input an encapsulation of a key C and a secret-key sk, and outputs a key K.

Obviously if the scheme is to be useful we require that, with overwhelming probability, the scheme is sound, i.e. for almost all $(pk, sk) = KEM.Gen(1^\lambda)$ and almost all $(K, C) = KEM.Encap(pk)$ we have that $K = KEM.Decap(C, sk)$. We also assume that the range of possible keys K is some set of fixed length binary strings, $\{0,1\}^{KEM.Keylen(\lambda)}$.

There are two ways to approach provable security. We choose to approach the subject from the complexity theoretic/asymptotic point of view and suggest that a scheme is secure if the probability of breaking that scheme is negligible as a function of the security parameter.

Definition 1. *A function f is said to be negligible if, for all polynomials p, there exists a constant N_p such that*

$$f(x) \leq \frac{1}{p(x)} \text{ for all } x \geq N_p \, .$$

A KEM is considered secure if there exists no attacker with a significant advantage in winning the following game played against a mythical system.

1. The system generates a public/secret key-pair $(pk, sk) = KEM.Gen(1^\lambda)$ and passes pk to the attacker.
2. The attacker runs until it is ready to receive a challenge encapsulation pair. During this time the attacker may repeatedly query a decapsulation oracle to find the key associated with any encapsulation.
3. The system prepares a challenge encapsulated key-pair as follows:
   (a) The system generates a valid encapsulated key-pair $(K_0, C) = KEM.Encap(pk)$.
   (b) The system selects an alternate key K_1 chosen uniformly at random from the set $\{0,1\}^{KEM.Keylen(\lambda)}$.
   (c) The system selects a bit σ uniformly at random from $\{0,1\}$.

   The system then passes (K_σ, C) to the attacker.
4. The attacker is allowed to run until it outputs a guess σ' for σ. During this time the attacker may repeatedly query a decapsulation oracle to find the key associated with any encapsulation except the challenge encapsulation C.

The attacker is said to win this game if $\sigma' = \sigma$. We define an attacker's advantage Adv to be

$$Pr[\sigma' = \sigma] - 1/2$$

and the total advantage of the KEM to be the maximum advantage of any polynomial time attacker. If the total advantage of a KEM is negligible (as a function of λ) then the KEM is said to be IND-CCA2 secure.

A KEM is only useful when coupled with a *DEM* (a *data encapsulation mechanism*) to form a hybrid public-key encryption scheme. A DEM is a symmetric algorithm that takes a message and a key, and encrypts the message under that key. However the security of the KEM is unaffected by the security properties of the DEM (and vice-versa) so we will only consider the security properties of the KEM. For further details of hybrid constructions using KEMs and DEMs, and their security properties, the reader is referred to [3].

3 Constructing KEMs from trapdoor one-way functions

We start by showing that the construction ideas used in RSA KEM [7] generalise to almost all one-way public-key encryption schemes. Consider an encryption scheme $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ where

– $\mathcal{G}$ is the key-generation algorithm which takes as input a security parameter 1^λ and outputs a public/secret key-pair (pk, sk),
– $\mathcal{E}$ is the encryption algorithm which takes as input a message $m \in \mathcal{M}$ and the public-key pk and outputs a ciphertext $C \in \mathcal{C}$,
– $\mathcal{D}$ is the decryption algorithm which takes as input a ciphertext $C \in \mathcal{C}$ and the secret-key sk and outputs either a message $m \in \mathcal{M}$ or the error symbol $\perp$.

Definition 2. *A public-key encryption scheme $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ is said to be one-way if the probability that a polynomial time attacker $\mathcal{A}$ can invert a randomly generated ciphertext C is negligible as a function of λ. Such a cryptosystem is often said to be secure in the OW-CPA model*[1].

The construction of a KEM derived from an encryption scheme $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ is given in Table 1. This construction uses a key derivation function KDF. This function is intended to do more than simply format the random number correctly as a key: it is meant to remove algebraic relations between inputs. It is usually constructed from a hash function and will be modelled as a random oracle.

We begin our analysis of this construction by restricting ourselves to the case where the encryption function acts as a permutation on the message space $\mathcal{M}$ (hence $\mathcal{M} = \mathcal{C}$ and the encryption function is necessarily deterministic in nature).

[1] OW for "one-way" and CPA for "chosen plaintext attack". The term "chosen plaintext attack" is used because the attacker is not allowed to make decryption queries.

Table 1. A KEM derived from a deterministic encryption scheme

– Key generation is given by the key generation algorithm of the public-key encryption scheme (i.e. $KEM.Gen = \mathcal{G}$).
– Encapsulation is given by:
  1. Generate an element $r \in \mathcal{M}$ uniformly at random.
  2. Set $C := \mathcal{E}(r, pk)$.
  3. Set $K := KDF(r)$.
  4. Output (K, C).
– Decapsulation of an encapsulation C is given by:
  1. Set $r := \mathcal{D}(C, sk)$.
  2. Set $K := KDF(r)$.
  3. Output K.

Theorem 1. *Suppose $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ is a deterministic public-key encryption scheme secure in the OW-CPA model such that, for all public-keys pk, the encryption function $\mathcal{E}(\cdot, pk)$ acts as a permutation on the message space. Then the KEM derived from $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ is, in the random oracle model, IND-CCA2 secure.*

Proof. We intend to show that if there exists an attacker $\mathcal{A}$ that successfully breaks the scheme, i.e. breaks the scheme with non-negligible advantage, then there exists an algorithm that inverts the one-way encryption function with some non-negligible probability. In order to do this we need to slightly tweak the environment in which we run the attacker $\mathcal{A}$.

Let Game 1 be the normal IND-CCA2 game, as described in Section 2. Let Game 2 be similar to Game 1 but with the challenge key-pair (K_σ, C) chosen at the start (although not given to the attacker). If the attacker $\mathcal{A}$ queries the decryption oracle at any point with the ciphertext C then the decryption oracle responds with the error symbol $\perp$. Notice that the only difference between the two games occurs if $\mathcal{A}$ queries the decryption oracle with the ciphertext C before it has been given the challenge key-pair (K_σ, C). In order to analyse the effects of this change of environment on the advantage of $\mathcal{A}$ we need the following simple lemma.

Lemma 1. *If A, B and E are events in some probability space and $Pr[A \mid \neg E] = Pr[B \mid \neg E]$ then $|Pr[A] - Pr[B]| \leq Pr[E]$.*

Now, let A be the event that $\mathcal{A}$ wins in Game 1, i.e. that it correctly identifies σ from (K_σ, C), and let B be the event that $\mathcal{A}$ wins in Game 2. Let E be the event that $\mathcal{A}$ asks for the decryption of C before the challenge is issued. Note that if E does not occur then $\mathcal{A}$ receives exactly the same information from the oracles in both games and so $Pr[A|\neg E] = Pr[B|\neg E]$. Hence

$$|Pr[A] - Pr[B]| \leq Pr[E].$$

Suppose that $\mathcal{A}$ makes at most q_D decryption queries. Since C is chosen uniformly at random from the set of possible ciphertexts and $\mathcal{A}$ has no knowledge of C before being issued with the challenge ciphertext, we have that

$$Pr[E] \leq \frac{q_D}{|\mathcal{M}|}.$$

Since the underlying encryption scheme is one-way, $1/|\mathcal{M}|$ must be negligible as a function of λ. Hence $Pr[B]$ is only negligibly less than $Pr[A]$. This leads to:

Lemma 2. *If there exists an attacker for a KEM with non-negligible advantage in Game 1 then there exists an attacker for the same KEM with non-negligible advantage in Game 2.*

We may now proceed along more standard lines and prove that if there exists an attacker $\mathcal{A}$ that breaks the KEM in Game 2 with advantage Adv then we can construct an algorithm that inverts the one-way public-key encryption function with probability comparable to Adv. Consider the following algorithm:

1. Prepare two lists $KDFList$ and $DecList$, initially setting them both to be empty.
2. We receive a public-key pk from the system and pass this straight to $\mathcal{A}$. We also receive a challenge ciphertext C^* to invert which we do not yet pass to $\mathcal{A}$.
3. Allow $\mathcal{A}$ to run until it requests a challenge encapsulated key-pair. During this time $\mathcal{A}$ can repeatedly ask us to decapsulate any encapsulated key C or to evaluate the key-derivation function $KDF(\cdot)$ (modelled as a random oracle) on a given input. If $\mathcal{A}$ asks for the decapsulation of the ciphertext C then we perform the following steps:
   (a) Check to see if $C = C^*$. If so, output $\perp$ and halt.
   (b) Check to see if there exists an entry $(C, K) \in DecList$. If so, output K and halt.
   (c) Generate a key K uniformly at random from the set $\{0,1\}^{KEM.Keylen}$ and add (C, K) to $DecList$.
   (d) Output K.

   If $\mathcal{A}$ asks us to evaluate the key derivation function $KDF(\cdot)$ on the input X then we perform the following steps:
   (a) Check to see if there exists an entry of the form $(X, K) \in KDFList$. If so, output K and halt.
   (b) Generate a key K uniformly at random from the set $\{0,1\}^{KEM.Keylen}$.
   (c) Check to see if $X \in \mathcal{M}$. If not, add (X, K) to $KDFList$, output K and halt.
   (d) Let $C = \mathcal{E}(X, pk)$. If $C = C^*$ then output X as the inverse of C^* and terminate the entire program.
   (e) Check to see if there exists an entry of the form $(C, K') \in DecList$. If so, add (X, K') to $KDFList$, output K' and halt.
   (f) Add (C, K) to $DecList$, (X, K) to $KDFList$, and output K.

4. In order to construct a challenge key-pair we generate a key K^* uniformly at random from the set $\{0,1\}^{KEM.Keylen}$. We pass the pair (K^*, C^*) to $\mathcal{A}$ as the challenge.
5. Allow $\mathcal{A}$ to run until it outputs a bit σ'. During this time we answer all queries $\mathcal{A}$ makes in exactly the same way as before.
6. Lastly, if the program has not yet terminated then we generate a guess X uniformly at random from $\mathcal{M}$ and output X as the inverse of C^*.

We can bound the probability of success of this algorithm in inverting the challenge ciphertext C^* in terms of the probability of success of the attacker $\mathcal{A}$ in winning Game 2. Let F be the event that $\mathcal{A}$, at some point, queries the key derivation function oracle with the value $X^* = \mathcal{D}(C^*, sk)$. The probability of inverting C^* is given by:

$$Pr[Output\ X^*] = Pr[Output\ X^*|F] \cdot Pr[F] + Pr[Output\ X^*|\neg F] \cdot Pr[\neg F].$$

Trivially we can see that

$$Pr[Output\ X^*|F] = 1.$$

It therefore suffices to bound $Pr[F]$. If F does not occur then the attacker can have no knowledge of $KDF(X^*) = KEM.Decap(C^*, sk)$, because $KDF(\cdot)$ is a random oracle; hence its advantage in that situation is 0. It is easy to show that, since $\mathcal{A}$ has advantage Adv, the probability that F occurs is at least Adv. Therefore,

$$Pr[OutputX^*] \geq Pr[F] \geq Adv$$

In particular, if $\mathcal{A}$ has a non-negligible advantage (in Game 2) then there exists an algorithm that inverts the one-way function with non-negligible probability. Putting this all together we get:

Lemma 3. *If there exists an attacker for the KEM with non-negligible advantage Adv then there exists an algorithm that inverts the underlying encryption scheme with probability at least*

$$Adv - q_D/|\mathcal{M}|\,.$$

On the other hand, if the underlying encryption scheme is one-way then the KEM is IND-CCA2 secure.

Hence Theorem 1 holds. □

Now suppose that $\mathcal{E}(\cdot, pk)$ is not a permutation (i.e. $\mathcal{M} \neq \mathcal{C}$) but instead a deterministic function, i.e. $\mathcal{E}(\mathcal{M}, pk) \subseteq \mathcal{C}$. In this situation a decryption query always reveals whether a ciphertext lies in $\mathcal{E}(\mathcal{M}, pk)$ or not. If such queries reveal information about the secret key then the above construction will be weak even if the underlying encryption scheme is one-way.

Definition 3. *A public-key encryption scheme $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ is said to be secure in the OW-CPA$^+$ model if it is one-way even when the attacker has access to an oracle that decides whether a ciphertext C lies in $\mathcal{E}(\mathcal{M}, pk)$ or not.*

This follows an idea first proposed in a paper by Joye, Quisquater and Yung [4]. Their paper attacks an early version of EPOC-2, a hybrid cipher based on the Okamoto-Uchiyama cryptosystem [6]. The paper proves that, whilst the Okamoto-Uchiyama cryptosystem is provably secure in the OW-CPA model, it is insecure in the OW-CPA$^+$ model. They were able to use this fact to construct an attack on EPOC-2.

Given a deterministic public-key encryption scheme that is secure in the OW-CPA$^+$ model, the construction given in Table 1 can be used to construct a secure KEM.

Theorem 2. *Suppose $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ is a deterministic public-key encryption scheme that is secure in the OW-CPA$^+$ model. Then the KEM derived from $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ is, in the random oracle model, IND-CCA2 secure.*

Sketch Proof The proof is very similar to the proof of Theorem 1. The only difference is that, when we are running the algorithm that breaks the one-way encryption scheme, we answer decryption queries from $\mathcal{A}$ by first checking whether that ciphertext has a valid decryption. If the ciphertext has no valid decryption then we return $\perp$; otherwise we simulate decryptions as before. □

4 Constructing KEMs from probabilistic encryption schemes

We now also open ourselves up to the possibility that the encryption algorithm is not deterministic but probabilistic. The first thing to note is that our current construction is not sufficient in the case of probabilistic encryption. Even if an encryption scheme is secure in the OW-CPA$^+$ model, it might still be possible to find, for some ciphertext C, a ciphertext $C' \neq C$ such that $\mathcal{D}(C', sk) = \mathcal{D}(C, sk)$. This would be sufficient to break the KEM in the adaptive chosen ciphertext model. Hence we propose a new construction, given in Table 2, based on a probabilistic encryption scheme $(\mathcal{G}, \mathcal{E}, \mathcal{D})$.

In order to prove the security of this scheme we will require access to an oracle that decides whether a given ciphertext C is an encryption of a given message m or not. Such oracles are rarely found in practice and it would be unusual for one to be included in a standard attack model/scenario. However there are several ways in which such an oracle could be made available to an attacker. In particular, it might be possible to construct such an oracle because of the nature of the underlying intractability assumption (see, for example, the gap problems [5]). Alternatively it might be possible, in the random oracle model, to simulate such an oracle using on the previous queries the attacker has made. A third option would be to construct the KEM from an encryption scheme that is secure even when the attacker has access to a decryption oracle (the OW-CCA2

Table 2. A KEM derived from a probabilistic encryption scheme

- Key generation for the KEM is provided by the key generation algorithm for the underlying scheme, $\mathcal{G}$.
- Encapsulation is provided by the following algorithm.
  1. Generate an element r uniformly at random from the set $\mathcal{M}$ (i.e. $KEM.Gen = \mathcal{G}$).
  2. Set $C := \mathcal{E}(r, pk)$.
  3. Set $K := KDF(\bar{r}||C)$, where $\bar{r}$ is a fixed length representation of the element $r \in \mathcal{M}$.
  4. Output (K, C).
- Decapsulation of an encapsulation C is given by the following algorithm.
  1. Set $r := \mathcal{D}(C, sk)$.
  2. Set $K := KDF(\bar{r}||C)$ where $\bar{r}$ is a fixed length representation of the element r.
  3. Output K.

model [1]). An encryption scheme that is secure in the OW-CCA2 model will certainly be secure in the OW-CPA$^+$ model even when the attacker has access to an oracle that decides if C is an encryption of m.

Theorem 3. *Suppose $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ is a public-key encryption scheme that is secure in the OW-CPA$^+$ model, even when the attacker has access to an oracle that checks whether a given ciphertext C (different from the challenge ciphertext) is an encryption of a given message m or not. Then the KEM derived from $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ is, in the random oracle model, IND-CCA2 secure.*

Sketch Proof See Appendix A.

5 Constructing KEMs from key-agreement protocols

One can not only build KEMs from encryption algorithms but also from simple key-agreement protocols. This should not be surprising considering the similarities between the two types of protocol! For our purposes we define a key-agreement scheme to be a triple of algorithms $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ where

- $\mathcal{G}$ is a parameter generation algorithm which takes as input a security parameter 1^λ and outputs a public/secret key-pair (pk, sk) that will be used to generate a shared key.
- $\mathcal{E}$ is a deterministic "encryption" algorithm which takes as input a public-key pk and a fixed length random seed $r \in \mathcal{R}$, and outputs a key-pair (K, C), where K is a key and C is an encryption of that key.
- $\mathcal{D}$ is a "decryption" algorithm which takes a ciphertext C and a secret key sk, and outputs a key K (or the error symbol $\perp$).

Notice that these definitions are very similar to the definition of a KEM itself. This is not accidental! However, whilst a KEM and a simple key-agreement protocol are functionally very similar, they have different security requirements. In particular a key-agreement protocol is designed to protect a key for a long period of time, whereas a KEM is designed only to protect a key that is used once. This means we can use a very weak notion of security for the underlying key-agreement protocol and still get full security in the KEM.

We will define the security of a key-agreement protocol as follows.

Definition 4. *A key-agreement protocol* $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ *is said to be secure in the OW-CPA model if, given a randomly generated encapsulation* C*, the probability that a polynomial time attacker* $\mathcal{A}$ *can find the corresponding key* $K = \mathcal{D}(C, sk)$ *is negligible as a function of* λ.

If the scheme is secure even when an attacker has access to an oracle that checks whether a given encapsulation C *is valid (or whether* $\mathcal{D}(C, sk) = \perp$*), then the scheme is said to be secure in the OW-CPA*$^+$ *model.*

It will also be necessary to allow the attacker access to an oracle that determines whether, for a given key K and ciphertext C, $K = \mathcal{D}(C, sk)$ or not. Once again we choose to leave this out of the formal attack model, partly to respect the model defined by Joye *et al.* and partly because this oracle can usually either be simulated in the random oracle model or be made available to the attacker owing to the nature of intractability assumption the scheme is based on (as is the case for schemes based on the gap Diffie-Hellman assumption [4]).

A method of deriving a KEM from a key-agreement protocol $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ is given in Table 3.

Table 3. A KEM derived from a key-agreement protocol

- Key generation is given by the key generation algorithm $\mathcal{G}$ of the key-agreement protocol.
- The encapsulation algorithm is given by:
  1. Generate an element r uniformly at random from the set $\mathcal{R}$.
  2. Set $(K_{raw}, C) := \mathcal{E}(r, pk)$.
  3. Set $K := KDF(\bar{K}_{raw}||C)$ where $\bar{K}_{raw}$ is a fixed length binary representation of K_{raw}.
  4. Output (K, C).
- The decapsulation algorithm, for a ciphertext C, is given by:
  1. Set $K_{raw} := \mathcal{D}(C, sk)$.
  2. Set $K := KDF(\bar{K}_{raw}||C)$ where $\bar{K}_{raw}$ is a fixed length binary representation of K_{raw}.
  3. Output K.

Theorem 4. *Suppose that $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ is a key-agreement protocol that is secure in the OW-CPA$^+$ model, even when the attacker has access to an oracle that decides whether a given encryption C (different from the challenge encapsulation) is an encryption of a given key K_{raw} or not. Then the KEM derived from $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ is, in the random oracle model, IND-CCA2 secure.*

The proof of this theorem is similar to that of Theorem 3.

6 Relaxing the security requirement

So far we have required our KEMs to be derived from lower level schemes that are secure in the OW-CPA$^+$ model. We may relax this requirement to the need for the lower level scheme to be secure in the OW-CPA model if we are prepared to reduce the efficiency of the KEM. The technique we use is based on the masking approach used by PSEC-KEM [7]. We will consider constructing a KEM using each of the types of lower level primitive we have already considered (a deterministic encryption scheme, a probabilistic encryption scheme and a simple key-agreement protocol), starting with a deterministic encryption scheme.

Table 4 gives a construction for a KEM from a deterministic encryption scheme $(\mathcal{G}, \mathcal{E}, \mathcal{D})$. In the construction $Hash$ is a hash function and MGF is a mask generating function. A mask generating function is similar to a key derivation function, in fact the same constructions are used for both, but a mask generating function is used to create a bit string that is used to mask a data value. We will model these function as random oracles. We also use a "smoothing function" $\phi : \{0,1\}^{n_2} \to \mathcal{M}$, where $\mathcal{M}$ is the message space of the encryption scheme $(\mathcal{G}, \mathcal{E}, \mathcal{D})$. This function must have the property that for any $r \in \{0,1\}^{n_2}$ and $m \in \mathcal{M}$ we have

$$Pr[\phi(r) = m] = \frac{1}{|\mathcal{M}|}.$$

For security, it is necessary that n is suitably large. Certainly $n \geq \lambda$ would be sufficient. Of the other lengths, n_1 should equal $KEM.Keylen$ and n_2 merely has to be large enough so that there exists a function ϕ which is suitably smooth.

Theorem 5. *Suppose $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ is a deterministic public-key encryption scheme secure in OW-CPA model. Then the KEM derived from $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ is, in the random oracle model, IND-CCA2 secure.*

Sketch Proof of Theorem 5 See Appendix B. □

This construction is not sufficient for use with a simple key-agreement protocol, because it might be possible to find two ciphertexts (for the key-agreement protocol) that decrypt to give the same raw key. However we can avoid this problem by changing step 6 of the encapsulation algorithm to:

6. Set $C_2 := r \oplus MGF(\bar{X}||C_1)$, where $\bar{X}$ is some fixed length representation of X.

and step 3 of the decapsulation algorithm to:

Table 4. A KEM derived from a OW-CPA secure encryption scheme

– Key-generation is given by $\mathcal{G}$, i.e. $KEM.Gen = \mathcal{G}$.
– Encapsulation is given by:
  1. Generate a suitably large bit-string $r \in \{0,1\}^n$.
  2. Set $R := Hash(r)$.
  3. Split R into two strings $K \in \{0,1\}^{n_1}$ and $R' \in \{0,1\}^{n_2}$ where $R = K||R'$.
  4. Set $X := \phi(R')$.
  5. Set $C_1 := \mathcal{E}(X, pk)$.
  6. Set $C_2 := r \oplus MGF(X)$.
  7. Set $C = (C_1, C_2)$.
  8. Output (K, C).
– Decapsulation is given by:
  1. Parse C as (C_1, C_2).
  2. Set $X := \mathcal{D}(C_1, sk)$.
  3. Set $r = C_2 \oplus MGF(X)$.
  4. Set $R = Hash(r)$.
  5. Split R into two strings $K \in \{0,1\}^{n_1}$ and $R' \in \{0,1\}^{n_2}$ where $R = K||R'$.
  6. Check that $C_1 = \mathcal{E}(\phi(R'), pk)$. If not, output $\perp$ and halt.
  7. Output K.

3. Set $r = C_2 \oplus MGF(\bar{X}||C_1)$, where $\bar{X}$ is some fixed length representation of X.

Theorem 6. *Suppose $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ is a simple key-agreement scheme (as defined in Section 5) secure in the OW-CPA model. Then the KEM derived from $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ is, in the random oracle model, IND-CCA2 secure.*

This construction will still not work with probabilistic constructions as the same message is unlikely to encrypt to the same ciphertext, so the decapsulation process will reject almost all ciphertexts. However we can alter the decryption process in order to avoid this problem by changing step 6 of the decapsulation process to:

6. Check that $\phi(R') = \mathcal{D}(C_1, sk)$. If not, output $\perp$ and halt.

Theorem 7. *Suppose $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ is a probabilistic public-key encryption scheme secure in the OW-CPA model, even when the attacker has access to an oracle that checks whether a given ciphertext C (that is not equal to the challenge ciphertext) is an encryption of a given message m or not. Then the KEM derived from $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ is, in the random oracle model, IND-CCA2 secure.*

The proofs of these theorems are similar to that of Theorem 5.

7 Conclusion

This paper has provided generic constructions for key-encapsulation mechanisms based on some very simple cryptographic protocols. Indeed, these results mean that a KEM can be constructed from almost any cryptographic problem. The abundance of KEMs and their variety serve to reinforce the idea that the KEM-DEM method of constructing a public-key encryption scheme is a very sound model. Indeed the ease with which one can construct secure KEMs, and hence secure hybrid encryption schemes, suggests that this is an area with enormous practical potential.

Acknowledgements

I would like to thank Victor Shoup, Louis Granboulan and Kenny Paterson for some very useful discussions in this area. As always the help of Christine Swart has been invaluable.

References

1. M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In *Advances in Cryptology – Crypto 98*, LNCS 1462, pages 26–45. Springer-Verlag, 1998.
2. D. Boneh, A. Joux, and A. Nguyen. Why textbook ElGamal and RSA encryption are insecure. In *Advances in Cryptology – Aisacrypt 2000*, LNCS 1976, pages 30–43. Springer-Verlag, 2000.
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. M. Joye, J. Quisquater, and M. Yung. On the power of misbehaving adversaries and security analysis of the original EPOC. In *Topics in Cryptography – CT-RSA 2001*, LNCS 2020, pages 208–222. Springer-Verlag, 2001.
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. T. Okamoto and S. Uchiyama. A new public-key cryptosystem as secure as factoring. In *Advances in Cryptology – Eurocrypt 98*, LNCS 1403, pages 308–318. Springer-Verlag, 1998.
7. V. Shoup. A proposal for the ISO standard for public-key encryption (version 2.0). Available from `http://shoup.net/`, 2001.

A Sketch Proof of Theorem 3

This proof runs along very similar lines to the proof of Theorem 1. Suppose there exists an attacker $\mathcal{A}$ which can distinguish a valid encapsulated key-pair from an invalid encapsulated key-pair in the IND-CCA2 environment described in Section 2.

Let Game 1 be the normal IND-CCA2 environment for $\mathcal{A}$ (as described in Section 2). Let Game 2 be similar to Game 1 but with the challenge key-pair (K^*_σ, C^*) chosen at the start. If $\mathcal{A}$ queries the decryption oracle at any time with the ciphertext C^* then the decryption oracle responds with $\perp$.

Let A be the event that $\mathcal{A}$ wins (i.e. correctly identifies whether (K^*_σ, C^*) is a valid key-pair or not) in Game 1. Let B be the event that $\mathcal{A}$ wins in Game 2. Then, exactly as in the proof of Theorem 1, we can show that

$$|Pr[A] - Pr[B]| \leq \frac{q_D}{|\mathcal{C}|},$$

where q_D is the number of decryption queries that $\mathcal{A}$ makes.

Hence, if there exists an attacker with non-negligible advantage in Game 1 then there exists an attacker with non-negligible advantage in Game 2.

Now we show that if there exists an attacker with advantage Adv in Game 2 then there exists an algorithm that inverts the encryption function with probability comparable to Adv. This is done is exactly the same manner as in Theorem 1, the only difference is how we simulate the oracles to which the attacker has access. Again we model the key derivation function $KDF(\cdot)$ as a random oracle, so we need to model both the decryption oracle and the random oracle representing the key derivation function. The simulators are described below:

If $\mathcal{A}$ asks for the decapsulation of an encapsulation C then we perform the following steps:

1. Check to see if $C = C^*$. If so, output $\perp$ and halt.
2. Check to see if there exists an entry $(C, K) \in DecList$, for some K. If so, output K and halt.
3. Check to see if $C \in \mathcal{E}(\mathcal{M}, pk)$ using the appropriate oracle. If not, output $\perp$ and halt.
4. Generate a new key K uniformly at random from the set $\{0,1\}^{KEM.Keylen}$ and add (C, K) to $DecList$.
5. Output K.

If $\mathcal{A}$ asks for the evaluation of $KDF(X)$ then we perform the following steps:

1. Check to see if there exists an entry $(X, K) \in KDFList$, for some K. If so, output K and halt.
2. Check to see if we can parse X as $\bar{r}||C$ for some representation $r \in \mathcal{M}$ and $C \in \mathcal{C}$. If not, generate a key K uniformly at random from the set $\{0,1\}^{KEM.Keylen}$, add (X, K) to $KDFList$, output K and halt.
3. Check to see if $C = C^*$. If so, generate a key K uniformly at random from the set $\{0,1\}^{KEM.Keylen}$, add (X, K) to $KDFList$, output K and halt.
4. Check to see if C is an encryption of the message r (using the appropriate oracle). If not, generate a key K uniformly at random from the set $\{0,1\}^{KEM.Keylen}$, add (X, K) to $KDFList$, output K and halt.
5. Check to see if there exists an entry (C, K) in $DecList$. If so, add (X, K) to $KDFList$, output K and halt.

6. Generate a key K uniformly at random from the set $\{0,1\}^{KEM.Keylen}$. Add (X,K) to $KDFList$, add (C,K) to $DecList$, output K and halt.

In the proof of Theorem 1, the simulators were designed to output the inverse of C^* directly, however this is not possible in this construction. Instead we must allow $\mathcal{A}$ to complete his program and then, if $KDFList$ contains an entry (X,K) such that $X = \bar{r}||C^*$ (for some r), we select such an entry uniformly at random and output r as the inverse of C^*. If no such entry exists then we select $r \in \mathcal{M}$ uniformly at random and output r as the inverse of C^*.

As before we can show that the probability that this algorithm outputs $r^* = \mathcal{D}(C^*, sk)$ is greater than or equal to the probability that the key derivation function is queried on the input $\bar{r^*}||C^*$. The probability that the key derivation function is queried on the input $\bar{r^*}||C^*$ is, in turn, greater than or equal to the advantage that $\mathcal{A}$ has in winning Game 2. So if the advantage of $\mathcal{A}$ is Adv then the probability that we correctly output the inverse of C^* is at least

$$\frac{1}{q_K}\left\{Adv - \frac{q_D}{|\mathcal{C}|}\right\}.$$

Of course, this assumes that our oracle cannot check if a given r is the decryption of C^* or not. If the oracle can decide this then the probability that the algorithm correctly inverts C^* is only negligibly less than the advantage of $\mathcal{A}$.

B Sketch Proof of Theorem 5

The proof of Theorem 5 is, again, very similar to the proof of Theorem 1 but several orders of magnitude more difficult. We will give a sketch of the proof here.

Assume that there exists an attacker $\mathcal{A}$ which breaks the KEM in the IND-CCA2 sense. We will use this to construct an algorithm that breaks the underlying cryptosystem. Assume that $\mathcal{A}$ makes at most q_D queries to the decapsulation oracle, q_M queries to the mask generating function oracle and q_H queries to the hash function oracle. In constructing the an algorithm that inverts the underlying cryptosystem we will be challenged to invert a random ciphertext C_1^* (we will use the superscript * to denote variables associated with the challenge ciphertext). For convenience we set

$$r^* = C_2^* \oplus \mathcal{D}(C_1^*, sk)$$

and note that this constrains the behaviour of $Hash$ on r^*.

We construct the challenge encapsulation pair (K^*, C^*) by selecting a random $KEM.Keylen$-bit integer K^* and a random n-bit integer C_2^* and setting $C^* = (C_1^*, C_2^*)$.

Again we need to tweak the environment that the attacker runs in, so that we may successfully simulate all the oracles that it has access to. Let Game 1 be the normal IND-CCA2 game. Let Game 2 be the game where the challenge encapsulation pair (K^*, C^*) is generated at the start of the algorithm, and if the attacker queries the decapsulation oracle on the input C^* then the decapsulation oracle responds with $\perp$. If A_1 is the event that the attacker wins in Game 1 and A_2 is the event that the attacker wins in Game 2 then, as before,

$$|Pr[A_1] - Pr[A_2]| \leq \frac{q_D}{2^n \cdot |\mathcal{C}|} \, .$$

Now, since we do not know $\mathcal{D}(C_1^*, sk)$, we will find it hard to simulate the decapsulation of encapsulations of the form (C_1^*, C_2). We can avoid this problem by refusing to decapsulate any encapsulation of this form. Let Game 3 be similar to Game 2, but with the decapsulation oracle will outputting $\perp$ whenever it is queried with an encapsulation of the form (C_1^*, C_2). Let A_3 be the event that the attacker wins in Game 3 and let E be the event that an encapsulation is submitted to the decapsulation oracle which would have different decapsulations in Game 2 and Game 3. Since Game 2 and Game 3 are identical if E does not occur we must have

$$|Pr[A_2] - Pr[A_3]| \leq Pr[E] \, .$$

Now E will only occur if the attacker $\mathcal{A}$ submits a ciphertext (C_1^*, C_2), with $C_2 \neq C_2^*$, for which the last n_2-bits of

$$Hash(C_2 \oplus \mathcal{D}(C_1^*, sk))$$

maps, under ϕ, to $\mathcal{D}(C_1^*, sk)$. Since $C_2 \oplus \mathcal{D}(C_1^*, sk) \neq r^*$ and $Hash$ is a random oracle, $Hash(C_2 \oplus \mathcal{D}(C_1^*, sk))$ will be a random bit-string and so ϕ will map the last n_2 bits of this onto a completely random element of $\mathcal{M}$. So

$$|Pr[A_3] - Pr[A_4]| \leq Pr[E] \leq \frac{q_D}{|\mathcal{M}|} \, .$$

We are now in a position to describe the simulators. We start by initialising four empty lists: $DecList$, $MaskList$, $MGFList$ and $HashList$. If the attacker requests the evaluation of the mask generating function MGF on the input X then the following steps are performed:

1. Check to see if there exists a pair $(X, Mask) \in MGFList$, for some value $Mask$. If so, output $Mask$ and halt.
2. Check to see if $X \in \mathcal{M}$. If not, generate $Mask$ uniformly at random from the set $\{0,1\}^n$, add $(X, Mask)$ to $MGFList$ and output $Mask$.
3. Check to see if $\mathcal{E}(X, pk) = C_1^*$. If so, output X as the inverse of C_1^* and terminate the entire algorithm.
4. Check to see if $(\mathcal{E}(X, pk), Mask) \in MaskList$ for some value of $Mask$. If so, output $Mask$ and halt.
5. Generate $Mask$ uniformly at random from the set $\{0,1\}^n$, add $(X, Mask)$ to $MGFList$, add $(\mathcal{E}(X, pk), Mask)$ to $MaskList$ and output $Mask$.

If the attacker requests the decapsulation of the encapsulation (C_1, C_2) then the following steps are performed:

1. Check to see if $(C_1, C_2, K) \in DecList$. If so, output K and halt.

2. Check to see if $C_1 = C_1^*$. If so, output $\perp$ and halt.
3. Check to see if $(C_1, Mask) \in MaskList$ for some value of $Mask$. If not, generate $Mask$ uniformly at random from the set $\{0,1\}^n$ and add $(C_1, Mask)$ to $MaskList$.
4. Set $r = C_2 \oplus Mask$.
5. Set $R = Hash(r)$.
6. Split R into two strings $K \in \{0,1\}^{n_1}$ and $R' \in \{0,1\}^{n_2}$ where $R = K||R'$.
7. Check to see if $C_1 = \mathcal{E}(\phi(R'), pk)$. If not, add $(C_1, C_2, \perp)$ to $DecList$, output $\perp$ and halt.
8. Add (C_1, C_2, K) to $DecList$, output K and halt.

If the attacker (or the decryption function) requests the evaluation of the hash function $Hash$ on the input r then the following steps are performed:

1. Check to see if $(r, R) \in HashList$ for some value of R. If so, output R and halt.
2. Otherwise, generate R uniformly at random from the set $\{0,1\}^{n_1+n_2}$, add (r, R) to $HashList$ and output R.

We use these simulators in the standard way, as shown in Section 3, to invert the ciphertext C_1^*. We therefore require that the simulators perfectly simulate the attacker's normal environment *up until the point where the inverse of C_1^* is found.* It is with some regret that we note that this is not the case at the moment, because when the hash function is queried on the input r^* the simulators will output a random bit string instead of the "proper" answer R^*. We must tweak the environment slightly to show that this does not make a significant difference.

Let Game 4 be similar to Game 3 but, if the hash function is evaluated on the input r^* *before* the mask generating function is evaluated on the input $\mathcal{D}(C_1^*, sk)$, then the hash function outputs an appropriately sized bit-string that has been generated at random. The simulators certainly perfectly simulate this environment. Let A_4 be the event that the attacker wins in Game 4 and let E be the event that the hash function is evaluated on the input r^* before the mask generating function is evaluated on the input $\mathcal{D}(C_1^*, sk)$. Since Game 3 and Game 4 are identical provided E does not occur, so

$$|Pr[A_3] - Pr[A_4]| \leq Pr[E] \, .$$

Now, since the mask generating function has not been evaluated on $\mathcal{D}(C_1^*, sk)$, the attacker can have no knowledge of $MGF(\mathcal{D}(C_1^*, sk)) = C_2^* \oplus r^*$. So, since C_2^* is known, the attacker can have no knowledge of r^* and so the only way that the hash function $Hash$ can be evaluated on r^* is by chance. Therefore,

$$|Pr[A_3] - Pr[A_4]| \leq Pr[E] \leq \frac{q_H + q_D}{2^n}$$

Now that we have now simulated the environment successfully, we can use standard techniques to show that the probability that we successfully invert the given ciphertext is at least the advantage of the attacker in Game 4. Hence the probability that we successfully invert the ciphertext in Game 1 is at least

$$Adv - \frac{q_H + q_D}{2^n} - \frac{q_D}{|\mathcal{M}|} - \frac{q_D}{2^n|\mathcal{C}|} \, .$$

So, if there exists an attacker for the KEM that has non-negligible advantage then there exists an algorithm that inverts the underlying encryption scheme with non-negligible advantage. Alternatively, if the underlying encryption scheme is one-way secure then the KEM will be IND-CCA2 secure.