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 key encapsulation mechanism (KEM) with a symmetric data encapsulation mechanism (DEM). A KEM is a probabilistic algorithm that produces a random symmetric key and an asymmetric encryption of that key. A DEM is a deterministic algorithm that takes a message and a symmetric key and encrypts the message under that key. Each of these components must satisfy its own security conditions if the overall scheme is to be secure. In this paper we describe generic constructions for provably secure KEMs based on weak encryption algorithms. We analyse the two most popular techniques for constructing a KEM and note that they are either overly complex or based on needlessly strong assumptions about the security of the underlying trapdoor function. Hence we propose two new, simple methods for constructing a KEM where the security of the KEM is based on weak assumptions about the underlying function. Lastly we propose a new KEM based on the Rabin function that is both efficient and secure, and is the first KEM to be proposed whose security depends upon the intractability of factoring.

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 [3] 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 [4]. 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.

This approach to the construction of hybrid ciphers has quickly become popular. Not only have several KEM schemes been proposed in the research literature [4, 8] but this approach has been adopted by the ISO standardisation body [13]. However KEMs are still proposed in an *ad hoc* fashion. Currently, if one wishes to propose a KEM based on one particular trapdoor problem then it is necessary to design such a KEM from scratch.

In this paper we examine generic methods for constructing KEMs from weak encryption schemes. We analyse the two methods for constructing a KEM based on existing schemes and show that either they require the underlying encryption scheme to have security properties which are stronger than they need to be or they are overly complex. We also provide two new generic construction methods which overcome these problems. Essentially this paper gives a toolbox to allow an algorithm designer to construct a KEM from almost any cryptographic problem. To demonstrate the power of the results we will also propose a new KEM, Rabin-KEM, that is as secure as factoring.

It should be noted that most of the results contained in this paper can be easily adapted to simple, "one-pass" key-agreement protocols like the Diffie-Hellman key agreement scheme [5].

2 The Security of a KEM

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, $\{0,1\}^{KEM.KeyLen(\lambda)}$.

We choose to approach provable security from an asymptotic/complexity theoretic 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 challenger.

1. The challenger 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 challenger prepares a challenge encapsulated key-pair as follows:
   (a) The challenger generates a valid encapsulated key-pair $(K_0, C) = KEM.Encap(pk)$.
   (b) The challenger selects an alternate key K_1 chosen uniformly at random from the set $\{0,1\}^{KEM.Gen(\lambda)}$.
   (c) The challenger selects a bit σ uniformly at random from $\{0,1\}$.
   The challenger 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

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

If the maximum advantage of any attacker against 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. In order for the overall hybrid encryption scheme to be secure, the KEM and the DEM must satisfy certain security properties. Happily these properties are independent (i.e. the security properties that a KEM must have are independent of the security properties of the DEM). For the overall encryption scheme to be IND-CCA2 secure (in the sense given below) the KEM, in particular, must be IND-CCA2 secure. For further details of hybrid constructions using KEMs and DEMs, and their security properties, the reader is referred to [4].

3 The Security of an Encryption Scheme

We will require formal definitions for an asymmetric encryption scheme. It will suit our purposes to draw a distinction between a deterministic and probabilistic encryption schemes as they present slightly different challenges to the KEM designer. We will start by considering deterministic encryption schemes.

Definition 2. *A deterministic encryption scheme is a triple* $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ *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* $\perp$*.*

The weakest notion of security for a deterministic encryption scheme is one-way security.

Definition 3. *A deterministic 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]*.*

[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.

A deterministic encryption scheme $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ is said to be secure in the OW-CPA+ model if the scheme is one-way even when the attacker has access to an oracle that, when given a ciphertext $C \in \mathcal{C}$, determines whether C is a valid ciphertext or not, i.e. whether C is the correct encryption of some message or not.

The idea of allowing an attacker access to an oracle that correctly determines if a ciphertext is valid was first used in a paper by Joye, Quisquater and Yung [7]. The paper used such an oracle to attack an early version of the EPOC-2 cipher.

For our purposes, a probabilistic encryption scheme will be viewed as a deterministic scheme whose encryption algorithm takes some random seed as an extra input.

Definition 4. *A probabilistic encryption scheme is a triple $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ 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 $m \in \mathcal{M}$, a random seed $r \in \mathcal{R}$ 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 $\perp$.*

To cement the idea that this is a probabilistic system we require that, for all public keys pk that can be obtained from the key generation algorithm with an input 1^λ and for all $m \in \mathcal{M}$ we have that

$$|\{r \in \mathcal{R} : \mathcal{E}(m, r, pk) = C\}| \leq \gamma(\lambda)/|\mathcal{R}|$$

where $C \in \mathcal{C}$ and $\gamma(\lambda)/|\mathcal{R}|$ is negligible as a function of λ.[2]

Analogous notions of OW-CPA and OW-CPA+ security can be defined for probabilistic encryption schemes. However, there is another issue that will affect our ability to design KEMs based on probabilistic encryption schemes - the need for a plaintext-ciphertext checking oracle.

Definition 5. *For a asymmetric encryption scheme $(\mathcal{G}, \mathcal{E}, \mathcal{D})$, a plaintext-ciphertext checking oracle is an oracle that, when given a pair $(m, C) \in \mathcal{M} \times \mathcal{C}$, correctly determines whether C is an encryption of m or not.*

[2] This condition basically states that for any public key and ciphertext, there cannot be two many choices for r that encrypt a message to that ciphertext.

Obviously, if $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ is a deterministic algorithm then there exists an efficient plaintext-ciphertext checking oracle, however the situation is more complicated for a probabilistic encryption scheme. There are several ways in which a plaintext-ciphertext checking oracle for a probabilistic encryption scheme can be made be available to all parties in a security proof. In particular, it might be possible to construct such an oracle because of the nature of underlying intractability assumption (such as in the case of an encryption scheme based on the gap Diffie-Hellman problem, see [10]). Alternatively, it might be possible to simulate such an oracle using, say, knowledge of the hash queries an attacker has made in the random oracle model [2].

4 Analysing RSA-KEM

We present a method to construct a KEM from almost all one-way public-key encryption schemes; this generalises the ideas used in RSA-KEM [13].

The construction of a KEM from a deterministic 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.

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 $x \in \mathcal{M}$ uniformly at random.
  2. Set $C := \mathcal{E}(x, pk)$.
  3. Set $K := KDF(x)$.
  4. Output (K, C).
- Decapsulation of an encapsulation C is given by:
  1. Set $x := \mathcal{D}(C, sk)$. If $x = \perp$ then output $\perp$ and halt.
  2. Set $K := KDF(x)$.
  3. Output K.

Theorem 1. *Suppose $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ is a deterministic asymmetric encryption scheme that is secure in the OW-CPA+ model. Then the KEM derived from $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ in Table 1 is, in the random oracle model, IND-CCA2 secure.*

The proof of this theorem is similar to the of Theorem 2.

This style of KEM can be easily extended to the case where the underlying encryption scheme is probabilistic and not deterministic. The construction is given in Table 2. Note that the encapsulation is included in the input to the key derivation function to prevent the attacker from breaking the scheme by finding a second ciphertext C' that decrypts to the same value as the challenge ciphertext C.

Table 2. A KEM derived from a probabilistic 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 provided by the following algorithm.
  1. Generate elements $x \in \mathcal{M}$ and $r \in \mathcal{R}$ uniformly at random.
  2. Set $C := \mathcal{E}(x, r, pk)$.
  3. Set $K := KDF(\bar{x}||C)$, where $\bar{x}$ is a fixed length representation of the element $x \in \mathcal{M}$.
  4. Output (K, C).
- Decapsulation of an encapsulation C is given by the following algorithm.
  1. Set $x := \mathcal{D}(C, sk)$. If $x = \perp$ then output $\perp$ and halt.
  2. Set $K := KDF(\bar{x}||C)$ where $\bar{x}$ is a fixed length representation of the element $x \in \mathcal{M}$.
  3. Output K.

Theorem 2. *Suppose $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ is a probabilistic asymmetric encryption scheme*

- *that is secure in the OW-CPA+ model, and*
- *for which there exists a plaintext-ciphertext checking oracle.*

Then the KEM derived from $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ in Table 2 is, in the random oracle model, IND-CCA2 secure.

Proof See Appendix A

5 Analysing PSEC-KEM

Obviously it would be advantageous if we were able to remove the reliance on these non-optimal security criteria, i.e. produce a generic method for constructing a KEM from an OW-CPA encryption scheme rather than from an OW-CPA+ encryption scheme and requiring a plaintext-ciphertext checking oracle. In this section, we present a method to construct a KEM from a OW-CPA encryption scheme; this generalise the ideas used in PSEC-KEM [13].

Table 3 gives a construction for a KEM from a (deterministic or probabilistic) asymmetric 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, hence care must be taken to ensure that the outputs of the hash function and the mask generating function are suitably independent. 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 Y' drawn uniformly at random from $\{0,1\}^{n_2}$ and $X \in \mathcal{M}$ we have

$$Pr[\phi(Y') = X] - \frac{1}{|\mathcal{M}|}$$

is negligible.

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 3. *Suppose $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ is an asymmetric encryption scheme that is secure in the OW-CPA model. Then the KEM derived from $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ in Table 3 is, in the random oracle model, IND-CCA2 secure.*

A sketch proof is given in Appendix D.

6 A New Construction for a Deterministic Encryption Scheme

Although the construction given in Section 5 (the generalisation of PSEC-KEM) is based on weak assumptions, it is very complex and there are many places where a weakness may be introduced in an implementation. We now propose a simpler construction for designing a KEM based on a deterministic encryption scheme with similarly weak security assumptions. In other words, we build a secure KEM from a deterministic encryption scheme that is secure in the OW-CPA model, as opposed to the

Table 3. 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 $y \in \{0,1\}^n$.
  2. Set $Y := Hash(y)$.
  3. Split Y into two strings $K \in \{0,1\}^{n_1}$ and $Y' \in \{0,1\}^{n_2}$ where $Y = K||Y'$.
  4. Set $X := \phi(Y')$.
  5. Set $C_1 := \mathcal{E}(X, pk)$. (If $\mathcal{E}$ is probabilistic then generate a random seed $r \in \mathcal{R}$ and set $C_1 := \mathcal{E}(X, r, pk)$.)
  6. Set $C_2 := y \oplus MGF(X)$.
  7. Set $C = (C_1, C_2)$.
  8. Output (K, C).

– Decapsulation of an encapsulation C is given by:
  1. Parse C as (C_1, C_2).
  2. Set $X := \mathcal{D}(C_1, sk)$. If $x = \perp$ then output $\perp$ and halt.
  3. Set $y = C_2 \oplus MGF(X)$.
  4. Set $Y = Hash(y)$.
  5. Split Y into two strings $K \in \{0,1\}^{n_1}$ and $Y' \in \{0,1\}^{n_2}$ where $Y = K||Y'$.
  6. Check that $\phi(Y') = X$. If not, output $\perp$ and halt.
  7. Output K.

OW-CPA+ model as in Section 4. The construction can be viewed as a simpler version of the REACT construction [11].

Table 4 gives a construction of a KEM based on a deterministic asymmetric encryption scheme $(\mathcal{G}, \mathcal{E}, \mathcal{D})$. The scheme makes use of a key derivation function *KDF* and a hash function *Hash*. These functions will be modelled as random oracles and so care must be taken that their outputs are suitably independent.

Theorem 4. *Suppose that $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ is a deterministic encryption algorithm that is secure in the OW-CPA model. Then the KEM derived from $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ in Table 4 is, in the random oracle model, IND-CCA2 secure.*

Proof See Appendix B

This construction also has the advantage that the decryption algorithm need not return a unique solution but need only return a small subset of the message space that includes the original message, as, with high probability, the original message will be the only message in the subset that hashes to give the correct value of C_2. We will make heavy use of this fact in the specification of Rabin-KEM (see Sect. 8).

Table 4. A KEM derived from an OW-CPA secure, deterministic 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 $x \in \mathcal{M}$.
  2. Set $C_1 := \mathcal{E}(x, pk)$.
  3. Set $C_2 := Hash(x)$.
  4. Set $C := (C_1, C_2)$.
  5. Set $K := KDF(x)$.
  6. Output (K, C).
- Decapsulation of an encapsulation C is given by:
  1. Parse C as (C_1, C_2).
  2. Set $x := \mathcal{D}(C_1, sk)$. If $x = \perp$ then output $\perp$ and halt.
  3. Check that $C_1 = \mathcal{E}(x, pk)$. If not, output $\perp$ and halt. (Note, this step may be ignored if there exists only one ciphertext associated with each message or if it is computationally infeasible for an attacker to find two ciphertexts that decrypt to the same value.)
  4. Check that $C_2 = Hash(x)$. If not, output $\perp$ and halt.
  5. Set $K := KDF(x)$.
  6. Output K.

7 A New Construction for a Probabilistic Encryption Scheme

Although the previous KEM construction can be generalised to be used with a probabilistic encryption scheme, the security proof still relies on the existence of a plaintext-ciphertext checking oracle (which is always easily constructed for a deterministic encryption algorithm). We now give a construction for a probabilistic encryption scheme, loosely based on the ideas of [6], that does not require a plaintext-ciphertext checking oracle. It is interesting to note, however, that this construction *cannot* be used for a deterministic scheme.

Table 5 gives the construction of a KEM based on a OW-CPA secure, probabilistic encryption scheme. Furthermore the proof of security for this construction does not require there to exist a plaintext-ciphertext checking oracle. The scheme makes use of a key derivation function *KDF* and a hash function *Hash*. These functions will be modelled as random oracles and so care must be taken that their outputs are suitably independent.

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

Proof See Appendix C

Table 5. A KEM derived from an OW-CPA secure, probabilistic 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 $x \in \mathcal{M}$.
  2. Set $r := Hash(x)$.
  3. Set $C := \mathcal{E}(x, r, pk)$.
  4. Set $K := KDF(x)$.
  5. Output (K, C).
- Decapsulation is given by:
  1. Set $x := \mathcal{D}(C, sk)$. If $x = \perp$ then output $\perp$ and halt.
  2. Set $r := Hash(x)$.
  3. Check that $\mathcal{E}(x, r, pk) = C$. If not, output $\perp$ and halt.
  4. Set $K := KDF(x)$.
  5. Output K.

8 Case study: Rabin-KEM

We demonstrate the power of these results by proposing a new KEM whose security is equivalent to factoring: Rabin-KEM. The Rabin-KEM construction will be based on the generic construction given in Sect. 6 and the Rabin trapdoor permutation [9, 12]. The algorithm is described in Table 6.

Theorem 6. *Providing the factoring problem is hard, Rabin-KEM is, in the random oracle model, IND-CCA2 secure.*

Proof It is well known that the Rabin trapdoor function is one-way providing that the factoring assumption is hard [12]. Therefore, given that the factoring problem is intractable, the given KEM is IND-CCA2 secure in the random oracle model by Theorem 4. □

This KEM is both and efficient and secure, being the first KEM ever proposed whose security depends on the assumption that factoring is intractable. Of course there is a chance that the decryption algorithm will fail, i.e. that $KEM.Decap(C, sk) = \perp$ even though C is actually a valid encapsulation of a key K. However this will only happen if there is a collision in the hash function, which, as we model the hash function as a random oracle, only happens with probability $2^{-Hash.Len}$ (where $Hash.Len$ is the length of the output of the hash function).

Table 6. Rabin-KEM

Key Generation On input of 1^λ for some integer $\lambda > 0$,

1. Randomly generate two distinct primes p and q of bit length λ.
2. Set $n := pq$.
3. Set $pk := (n)$ and $sk := (p, q)$.
4. Output (pk, sk).

Encapsulation On input of a public key PK,

1. Randomly generate an integer $x \in [0, n)$.
2. Set $C_1 := x^2 \bmod n$.
3. Set $C_2 := Hash(x)$.
4. Set $C := (C_1, C_2)$.
5. Set $K := KDF(x)$.
6. Output (K, C).

Decapsulation On input of an encapsulated key C and a secret key sk.

1. Parse C as (C_1, C_2).
2. Check that C_1 is a square modulo n. If not, output $\perp$ and halt.
3. Compute the four square roots x_1, x_2, x_3, x_4 of C_1 modulo n using the secret key sk.
4. If there exists no value $1 \leq i \leq 4$ such that $Hash(x_i) = C_2$ then output $\perp$ and halt.
5. If there exists more than one value $1 \leq i \leq 4$ such that $Hash(x_i) = C_2$ then output $\perp$ and halt.
6. Let x be the unique square root of C_1 modulo n for which $Hash(x) = C_2$.
7. Set $K := KDF(x)$.
8. Output K

9 Conclusion

This paper has provided four generic constructions for key encapsulation mechanisms (KEMs): two generalisations of existing KEMs and two new KEMs. These results show that KEMs can be constructed from almost any trapdoor function. We also proposed a new KEM: Rabin-KEM. This is a new fast, secure KEM based on the intractability of factoring large numbers.

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 H. Krawczyk, editor, *Advances in Cryptology – Crypto '98*, LNCS 1462, pages 26–45. Springer-Verlag, 1998.
2. 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.
3. D. Boneh, A. Joux, and A. Nguyen. Why textbook ElGamal and RSA encryption are insecure. In T. Okamoto, editor, *Advances in Cryptology – Asiacrypt 2000*, LNCS 1976, pages 30–43. Springer-Verlag, 2000.
4. R. Cramer and V. Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. Available from `http://shoup.net/`, 2002.
5. W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, 1976.
6. E. Fujisaki and T. Okamoto. How to enhance the security of public-key encryption at minimal cost. In H. Imai and Y. Zheng, editors, *Public Key Cryptography*, LNCS 1560, pages 53–68. Springer-Verlag, 1999.
7. M. Joye, J. Quisquater, and M. Yung. On the power of misbehaving adversaries and security analysis of the original EPOC. In D. Naccache, editor, *Topics in Cryptography – CT-RSA 2001*, LNCS 2020, pages 208–222. Springer-Verlag, 2001.
8. S. Lucks. A variant of the Cramer-Shoup cryptosystem for groups of unknown order. In Y. Zheng, editor, *Advances in Cryptology – Asiacrypt 2002*, LNCS 2501, pages 27–45. Springer-Verlag, 2002.
9. A. J. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1997.
10. T. Okamoto and D. Pointcheval. The gap problems: A new class of problems for the security of cryptographic schemes. In K. Kim, editor, *Public Key Cryptography*, LNCS 1992, pages 104–118. Springer-Verlag, 2001.
11. T. Okamoto and D. Pointcheval. REACT: Rapid enhanced-security asymmetric cryptosystem transform. In D. Naccache, editor, *Proceedings of CT-RSA 2001*, LNCS 2020, pages 159–175. Springer-Verlag, 2001.
12. M. O. Rabin. Digitalized signatures and public-key functions as intractable as factorization. Technical Report MIT/LCS/TR-212, MIT Laboratory for Computer Science, 1979.
13. V. Shoup. A proposal for the ISO standard for public-key encryption (version 2.0). Available from `http://shoup.net/`, 2001.

A Proof of Theorem 2

This is a simple result. We will use standard techniques to prove a more detailed result.

Theorem 7. *Let $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ be an encryption scheme and let KEM be the KEM derived from $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ using the construction described in Table 2. If there exist an attacker $\mathcal{A}$ that, in the random oracle model, breaks KEM in the IND-CCA2 model*

- *with advantage ϵ,*
- *in time t,*
- *makes at most q_D queries to the decapsulation oracle,*
- *and at most q_K queries to the random oracle that represents the key derivation function,*

then there exists an algorithm that inverts the underlying encryption function in the OW-RA model (and makes use of a plaintext-ciphertext checking oracle) with probability ϵ' and in time t' where

$$\epsilon' \geq \epsilon\,, \tag{1}$$

$$t' = t\,. \tag{2}$$

Proof We assume that there exists an attacker for the KEM and use this to construct an algorithm that can break the underlying encryption scheme. Suppose $\mathcal{A}$ is an attacker that breaks the KEM with the properties stated above.

Consider the following algorithm that takes as input a public-key pk for the underlying encryption scheme and a challenge ciphertext C^*. This algorithm makes use of two lists: $KDFList$ which stores the answers to queries made to the KDF oracle and $DecList$ which stores the answers to queries made to the decapsulation oracle.

1. Prepare two empty lists $KDFList$ and $DecList$.
2. Randomly generate a bit-string K^* of length $KEM.KeyLen$.
3. Pass the public key to pk to $\mathcal{A}$.
4. Allow $\mathcal{A}$ to run until it requests a challenge encapsulation. If the attacker requests the evaluation of the key derivation function KDF on an input z then the following steps are performed:
   (a) Check to see if $(z, K) \in KDFList$ for some value of K. If so, return K.
   (b) Check to see if z can be parsed as $\bar{x}||C$ for some fixed length representation of a message $x \in \mathcal{M}$ and a ciphertext $C \in \mathcal{C}$. If not, randomly generate an appropriately sized K, add (x, K) to $KDFList$ and return K.
   (c) Check to see if C is an encryption of x using the plaintext-ciphertext checking oracle. If not, randomly generate an appropriately sized K, add (x, K) to $KDFList$ and return K.
   (d) Randomly generate an appropriately sized K, add (x, K) to $KDFList$, add (C, K) to $DecList$ and return K.

   If the attacker requests the decapsulation of an encapsulation C then the following steps are performed:

(a) Check to see if $C = C^*$. If so, return $\perp$.
(b) Check to see if $(C, K) \in DecList$ for some value K. If so, return K.
(c) Check to see if C is a valid ciphertext or not (using the oracle provided by the RA model). If not, add $(C, \perp)$ to $DecList$ and return $\perp$.
(d) Otherwise randomly generate an appropriately sized K, add (C, K) to $DecList$ and return K.

5. When the attacker requests a challenge encapsulation, return (K^*, C^*)
6. Allow the attacker to run until it outputs a bit σ'. Answer all oracle queries as before.
7. Check to see if there exists a pair $(z, K) \in KDFList$ such that z can be decomposed as $\bar{x}||C^*$ where $\bar{x}$ is the fixed length representation of a message $x \in \mathcal{M}$ and C^* is an encryption of x (using the plaintext-ciphertext checking oracle). If so, output x and halt.
8. Otherwise randomly generate $x \in \mathcal{M}$. Output x and halt.

This algorithm perfectly simulates the attack environment for $\mathcal{A}$ up until the point where $\mathcal{A}$ queries the key derivation function on $\bar{x}||C^*$ where $\bar{x}$ is the fixed length representation of $\mathcal{D}(C^*, sk)$. Since $\mathcal{A}$ can have no advantage in breaking the KEM unless this event occurs, we can show that this event must occur with probability at least ϵ. However if this event occurs then we will successfully recover $\mathcal{D}(C^*, sk)$. Hence the theorem holds. □

B Proof of Theorem 4

We use standard techniques to prove the following, slightly more detailed result.

Theorem 8. *Let $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ be a deterministic encryption scheme and let KEM be the KEM derived from $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ using the construction described in Table 4. If there exists an attacker $\mathcal{A}$ that, in the random oracle model, breaks KEM in the IND-CCA2 model*

- *with advantage ϵ,*
- *in time t,*
- *and makes at most q_D decapsulation queries,*
- *at most q_H queries to the random oracle that represents the hash function,*
- *and at most q_K queries to the random oracle that represents the key derivation function,*

then there exists an algorithm that inverts the underlying encryption function with probability ϵ' and in time t' where

$$\epsilon' \geq \epsilon - \frac{q_D}{2^{Hash.Len}}(1 + \frac{1}{|\mathcal{C}|}) \,, \tag{3}$$

$$t' \leq t + (q_H + q_K + q_D)T \,, \tag{4}$$

where Hash.Len is the length of the output of the hash function Hash and T is the time taken to evaluate the encryption function $\mathcal{E}$.

Proof We assume that there exists an attacker for the KEM and use this to construct an algorithm that can break the underlying encryption scheme. Suppose $\mathcal{A}$ is an attacker that breaks the KEM with the properties stated above.

First we slightly change the environment that $\mathcal{A}$ operates in. Let game 1 be the game in which $\mathcal{A}$ attacks the KEM as described in the IND-CCA2 environment described in Sect. 3. Let game 2 be similar to game 1 except that

– the challenge encapsulation (K^*, C^*) is chosen at the beginning of the algorithm and if the attacker ever requests the decapsulation of $C^* = (C_1^*, C_2^*)$ then the decapsulation algorithm returns $\perp$,
– instead of allowing the attacker $\mathcal{A}$ access to the "real" decapsulation oracle, hash function oracle and KDF oracle we only allow $\mathcal{A}$ to have access to the "partially simulated" versions of these oracles described below.

The simulated oracles make use of two lists $HashList$ and $KDFList$, both of which are initially empty. If the attacker requests the evaluation of the hash function *Hash* on an input x then the following steps are performed:

1. If $(x, hash) \in HashList$ for some value of $hash$ then return $hash$.
2. If $x = \mathcal{D}(C_1^*, sk)$ then return $Hash(x)$.
3. Otherwise randomly generate an appropriately sized $hash$, add $(x, hash)$ to $HashList$ and return $hash$.

Hence the hash function is changed to a random function with the proviso that it must agree with the original hash function on the input $\mathcal{D}(C_1^*, sk)$. This simulation is equivalent to some random oracle and every random oracle can be represented by this simulation. Similarly, if the attacker requests the evaluation of the key derivation function *KDF* on an input x then the following steps are performed:

1. If $(x, K) \in \mathit{KDFList}$ for some value of K then return K.
2. If $x = \mathcal{D}(C_1^*, sk)$ then return $\mathit{KDF}(x)$.
3. Otherwise randomly generate an appropriately sized K, add (x, K) to $\mathit{KDFList}$ and return K.

If the attacker requests the evaluation of decapsulation function on the encapsulated key (C_1, C_2) then the following steps are performed:

1. If $C_1 = C_1^*$ then return $\perp$.
2. Check that there exists a unique $x \in \mathcal{M}$ such that $(x, C_2) \in \mathit{HashList}$ and $\mathcal{E}(x, pk) = C_1$. If not, return $\perp$.
3. Compute $K := \mathit{KDF}(x)$ using the KDF algorithm described above.
4. Return K.

To analyse the effects of only allowing $\mathcal{A}$ to have access to the simulated oracles we require the following simple lemma [1, 4].

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

Let A be the event that $\mathcal{A}$ succeeds in breaking the KEM with access to the real oracles and let B be the event that $\mathcal{A}$ succeeds in breaking the KEM with access to the simulated oracles. Let E be the event that either

1. $\mathcal{A}$ queries the decapsulation oracle on the challenge encapsulation before the challenge encapsulation is given to $\mathcal{A}$, or
2. $\mathcal{A}$ queries the decapsulation oracle on some encapsulation (C_1, C_2) where $\mathit{Hash}(\mathcal{D}(C_1, sk)) = C_2$ but $\mathcal{A}$ has not queried the hash function simulator on the input $\mathcal{D}(C_1, sk)$.

If E does not occur then $\mathcal{A}$ will receive the same responses to his queries regardless of whether it is querying the real oracles or the simulated oracles. Hence $Pr[A|\neg E] = Pr[B|\neg E]$.

Since the challenge ciphertext has to be chosen completely at random, the probability that E occurs because $\mathcal{A}$ queries the decapsulation oracle on the challenge encapsulation before it has been issued is bounded above by $q_D/2^{\mathit{Hash.Len}}|\mathcal{C}|$. Since the hash function is modelled as a random oracle, the probability that $\mathcal{A}$ queries the decapsulation oracle on some encapsulation (C_1, C_2) where $\mathit{Hash}(\mathcal{D}(C_1, sk)) = C_2$ but $\mathcal{A}$ has not queried the hash function Hash on the input $\mathcal{D}(C_1, sk)$ is at most $q_D/2^{\mathit{Hash.Len}}$. Hence the advantage of $\mathcal{A}$ in game 2 is least

$$\epsilon - \frac{q_D}{2^{\mathit{Hash.Len}}}(1 + \frac{1}{|\mathcal{C}|}) . \tag{5}$$

Let E' be the event that, in game 2, the attacker queries either the hash function simulator or the key derivation function oracle with the input $x^* = \mathcal{D}(C_1^*, sk)$. Since the attacker can have no knowledge of the whether $KDF(x^*) = K^*$ or not unless E' occurs we have that

$$Pr[E'] \geq \epsilon - \frac{q_D}{2^{Hash.Len}}(1 + \frac{1}{|\mathcal{C}|})\,. \tag{6}$$

Consider the following algorithm that takes as input a public key pk for the underlying encryption scheme and a challenge ciphertext C_1^*.

1. Prepare two empty lists $HashList$ and $KDFList$.
2. Generate random bit strings C_2^* of length $Hash.Len$ and K^* of length $KEM.KeyLen$. Set $C^* := (C_1^*, C_2^*)$.
3. Pass the public key pk to $\mathcal{A}$.
4. Allow the attacker $\mathcal{A}$ to run until it requests a challenge encapsulation. If the attacker requests the evaluation of the hash function $Hash$ on an input x then the following steps are performed:
   (a) If $(x, hash) \in HashList$ for some value of $hash$ then return $hash$.
   (b) Otherwise randomly generate an appropriately sized $hash$, add $(x, hash)$ to $HashList$ and return $hash$.

   If the attacker requests the evaluation of the KDF KDF on an input x then the following steps are performed:
   (a) If $(x, K) \in KDFList$ for some value of K then return K.
   (b) Otherwise randomly generate an appropriately sized K, add (x, K) to $KDFList$ and return K.

   If the attacker requests the evaluation of decapsulation function on the encapsulated key (C_1, C_2) then the following steps are performed:
   (a) If $C_1 = C_1^*$ then return $\perp$.
   (b) Check that there exists a unique $x \in \mathcal{M}$ such that $(x, C_2) \in HashList$ and $\mathcal{E}(x, pk) = C_1$. If not, return $\perp$.
   (c) Compute $K := KDF(x)$ using the simulator described above.
   (d) Return K.
5. When the attacker requests a challenge encapsulation pass the pair (K^*, C^*) to the attacker.
6. Allow the attacker to run until it outputs a bit σ'. Answer all oracle queries with the simulators described above.
7. Check to see if there exists some $(x, hash) \in HashLish$ or $(x, K) \in KDFList$ such that $\mathcal{E}(x, pk) = C_1^*$. If so, output x and halt.
8. Randomly generate $x \in \mathcal{M}$. Output x and halt.

This algorithm perfectly simulates the attack environment for the attacker $\mathcal{A}$ in game 2, up until the point where event E' occurs. However, if E' occurs then the above algorithm will correctly output $x^* = \mathcal{D}(C_1^*, sk)$. Hence the above algorithm will correctly invert a randomly generated ciphertext with probability at least

$$\epsilon - \frac{q_D}{2^{Hash.Len}}(1 + \frac{1}{|\mathcal{C}|}) \,. \tag{7}$$

This value is negligible providing ϵ is negligible, hence the KEM is secure in the IND-CCA2 model providing the underlying encryption scheme is secure in the OW-CPA model. □

C Proof of Theorem 5

Again, we prove a slightly more detailed result.

Theorem 9. *Let $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ be a probabilistic encryption scheme and let KEM be the KEM derived from $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ using the construction described in Table 5. If there exists an attacker $\mathcal{A}$ that, in the random oracle model, breaks KEM in the IND-CCA2 model*

- *with advantage ϵ,*
- *in time t,*
- *and makes at most q_D decapsulation queries,*
- *at most q_H queries to the random oracle that represents the hash function,*
- *and at most q_K queries to the random oracle that represents the key derivation function.*

then there exists an algorithm that inverts the underlying encryption function with probability ϵ' and in time t' where

$$\epsilon' \geq \frac{1}{q_D + q_H + q_K}(\epsilon - \frac{q_D}{|\mathcal{M}|} - \frac{\gamma q_D}{|\mathcal{R}|}) \,, \tag{8}$$

$$t' \approx t \,, \tag{9}$$

where γ is defined in Definition 4.

Proof The proof is similar to that given in Appendix B.

Let game 1 be the game in which $\mathcal{A}$ attacks the KEM as described in the IND-CCA2 environment described in Sect. 3. Let game 2 be similar to game 1 except that

– the challenge encapsulation (K^*, C^*) is chosen at the beginning of the algorithm and if the attacker ever requests the decapsulation of C^* then the decapsulation algorithm returns $\perp$,
– instead of allowing the attacker $\mathcal{A}$ access to the "real" decapsulation oracle, hash function oracle and KDF oracle we only allow $\mathcal{A}$ to have access to the "partially simulated" versions of these oracles described below.

We simulate the hash function oracle and the KDF oracle exactly as before, making use of two lists, $HashList$ and $KDFList$, both of which are initially empty. If the attacker requests the evaluation of the hash function $Hash$ on an input x then the following steps are performed:

1. If $(x, hash) \in HashList$ for some value of $hash$ then return $hash$.
2. If $x = \mathcal{D}(C^*, sk)$ then return $Hash(x)$.
3. Otherwise randomly generate an appropriately sized $hash$, add $(x, hash)$ to $HashList$ and return $hash$.

If the attacker requests the evaluation of the key derivation function KDF on an input x then the following steps are performed:

1. If $(x, K) \in KDFList$ for some value of K then return K.
2. If $x = \mathcal{D}(C^*, sk)$ then return $KDF(x)$.
3. Otherwise randomly generate an appropriately sized K, add (x, K) to $KDFList$ and return K.

If the attacker requests the evaluation of the decapsulation function on the encapsulated key C then the following steps are performed:

1. If $C = C^*$ then return $\perp$.
2. For each pair $(x, hash) \in HashList$, check whether $\mathcal{E}(x, hash, pk) = C$. If no such pair exists then return $\perp$.
3. If there exists such a pair $(x, hash)$ then run the simulator for the key derivation function on the input x to get a key K.
4. Return K.

As before, we note that game 1 and game 2 are identical except if either of the following events occur:

1. $\mathcal{A}$ queries the decapsulation oracle on the challenge encapsulation before the challenge encapsulation is given to $\mathcal{A}$, or
2. $\mathcal{A}$ queries the decapsulation oracle on some encapsulation C where $x = \mathcal{D}(C, sk)$ and $C = \mathcal{E}(x, Hash(x), pk)$ but $\mathcal{A}$ has not queried the hash function simulator on the input x.

The probability that the first event occurs is bounded above by $q_D/|\mathcal{M}|$ (as there exists $|\mathcal{M}|$ valid encapsulations). The probability that the second event occurs is bounded above by $q_D\gamma/|\mathcal{R}|$ (where γ is defined in Definition 4). Hence the advantage of $\mathcal{A}$ in game 2 is at least

$$\epsilon - q_D(\frac{1}{|\mathcal{M}|} + \frac{\gamma}{|\mathcal{R}|}) . \tag{10}$$

Let E' be the event that, in game 2, the attacker queries either the hash function simulator or the key derivation function oracle with the input $x^* = \mathcal{D}(C^*, sk)$. Again, we have

$$Pr[E'] \geq \epsilon - q_D(\frac{1}{|\mathcal{M}|} + \frac{\gamma}{|\mathcal{R}|}) . \tag{11}$$

Now, consider the following algorithm that takes as input a public key pk for the underlying encryption scheme and a challenge ciphertext C^* (which is the encryption of some randomly chosen message $x^* \in \mathcal{M}$).

1. Prepare two empty lists $HashList$ and $KDFList$.
2. Generate a random bit strings K^* of length $KEM.KeyLen$.
3. Pass the public key pk to $\mathcal{A}$.
4. Allow the attacker $\mathcal{A}$ to run until it requests a challenge encapsulation. If the attacker requests the evaluation of the hash function $Hash$ on an input x then the following steps are performed:
   (a) If $(x, hash) \in HashList$ for some value of $hash$ then return $hash$.
   (b) Otherwise randomly generate an appropriately sized $hash$, add $(x, hash)$ to $HashList$ and return $hash$.

   If the attacker requests the evaluation of the KDF KDF on an input x then the following steps are performed:
   (a) If $(x, K) \in KDFList$ for some value of K then return K.
   (b) Otherwise randomly generate an appropriately sized K, add (x, K) to $KDFList$ and return K.

   If the attacker requests the evaluation of decapsulation function on the encapsulated key C then the following steps are performed:
   (a) If $C = C^*$ then return $\perp$.
   (b) Check that there exists a unique $x \in \mathcal{M}$ such that $(x, hash) \in HashList$ and $\mathcal{E}(x, hash, pk) = C$ for some value of $hash$. If not, return $\perp$.
   (c) Run the simulator for the key derivation function on the input x to get a key K.
   (d) Return K.

5. When the attacker requests a challenge encapsulation pass the pair (K^*, C^*) to the attacker.
6. Allow the attacker to run until it outputs a bit σ'. Answer all oracle queries with the simulators described above.
7. Pick, uniformly at random, some value x from the set of x such that either $(x, hash) \in HashList$ or $(x, K) \in KDFList$. Output x as the inverse of C^*.

This algorithm perfectly simulates the environment for the attacker in game 2 up until the point in which E' occurs. However if E' occurs then the above correctly output x^* with probability $1/(q_D + q_H + q_K)$. Hence the above algorithm will correctly invert the encryption of a randomly generated message with probability at least

$$\frac{1}{q_D + q_H + q_K}(\epsilon - \frac{q_D}{|\mathcal{M}|} - \frac{\gamma q_D}{|\mathcal{R}|}). \qquad (12)$$

This value is negligible providing ϵ is negligible, hence the KEM is secure in the IND-CCA2 model providing the underlying encryption scheme is secure in the OW-CPA model. □

D Proof of Theorem 3

Again, we actually prove a slightly more detailed result. Suppose $\mathcal{A}$ is an attacker which breaks the KEM in the IND-CCA2 model, and that this attacker

– has advantage ϵ,
– makes at most q_D queries to the decapsulation oracle,
– makes at most q_M queries to the mask generating function oracle,
– and at most q_H queries to the hash function oracle.

We will use this to construct an algorithm that breaks the underlying cryptosystem. In constructing the 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 $C^* = (C_1^*, C_2^*)$.

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, using Lemma 1, we have

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

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 D, 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.