# A Proposal for an ISO Standard for Public Key Encryption (version 2.0)

Victor Shoup
*IBM Zurich Research Lab, Säumerstr. 4, 8803 Rüschlikon, Switzerland*
sho@zurich.ibm.com

September 17, 2001

**Abstract**

This document should be viewed less as a first draft of a standard for public-key encryption, and more as a proposal for what such a draft standard should contain. It is hoped that this proposal will serve as a basis for discussion, from which a consensus for a standard may be formed.

## Summary of Changes from version 1.1 (February 13, 2001) to version 2.0 (September 17, 2001)

- Some new and quite severe criticisms of the security of *ECIES* in certain modes of operation have been added. Specifically, it is shown that in certain modes of operation, the scheme is malleable in a very strong and quite non-trivial sense. See §15 and in particular §15.6.4.
- The implementation of "cofactor mode" for *ECIES* is now based on the notion of "compatible cofactor mode" in IEEE P1363a, and the "old cofactor mode" is no longer recommended. See §15 as well as §15.6.5.
- A new "primitive" called a *data encapsulation mechanism* has been introduced (see §4). This primitive was implicit in previous versions, but now it has been made explicit. This is mainly just a conceptual change. Note, however, that there is one slight change in the recommended implementation of the primitive (which we call *DEM1*); namely, the way in which the length of the label is formatted when passed to the MAC. This is done so as to align with IEEE P1363a. This change affects *ECIES*, as well as other hybrid schemes. See §10.
- A new section has been added that details all the differences between our recommended version of *ECIES* and that recommended in the IEEE P1363a standard. Note that our recommended version of *ECIES* is consistent with the IEEE P1363a version — the only differences between the two are that our recommended version does not allow several options and modes of operations allowed by the IEEE P1363a version. See §15.6.
- Two changes were made to *ACE-KEM* (formerly *ACE-Encrypt'*). First, the value of $\mathbf{v}$ is now encoded in the ciphertext using the group encoding function, rather than the partial encoding function as in previous versions. This was done mainly for aesthetic reasons. Second, the implementation of "cofactor mode" has been changed to align with the IEEE P1363a notion of "compatible cofactor mode." See §17.
- It is now recommended that all Diffie-Hellman-based schemes use prime order subgroups. This is done mainly for alignment with other standards, as well overall consistency.
- Following discussions in Santa Barbara, this report now recommends inclusion of *EPOC-2* in the standard, although this report does not yet include a detailed specification of *EPOC-2*.
- A new final section has been added that recommends the next steps to be taken toward an ISO standard. See §21.
- A number of names have been changed. Some of these name changes were suggested by working group members, while others were made to increase overall consistency.

| old name | new name |
|---|---|
| *ECIES'* | *ECIES-KEM* |
| *PSEC-2'* | *PSEC-KEM* |
| *ACE-Encrypt'* | *ACE-KEM* |
| *Simple RSA* | *RSA-KEM* |
| *EME-OAEP* | *OAEP-EME* |
| *XEME-OAEP+* | *OAEP+XEME* |

- A number of other, essentially aesthetic, changes.

## Summary of Changes from original version (February 13, 2001) to version 1.1 (May 29, 2001)

- The decryption algorithm for ACE-Encrypt$'$ has been slightly modified (see §17.3).
- Additionally, some minor errors — not affecting the descriptions of any algorithms — have been fixed.

## Contents

# 1 Introduction

## 1.1 Background

At its meeting on April 3-7 2000 in London, the ISO/IEC JTC 1/SC 27/WG 2 decided to put out a call for contributions for a proposed new project (NP 18033) on *encryption algorithms*. This call for contributions (document SC 27 N 2563) proposed four parts:

1. General
2. Asymmetric Ciphers
3. Block Ciphers
4. Stream Ciphers

The author of this document is currently the acting editor for the Asymmetric Ciphers part of the standard. This document deals exclusively with asymmetric ciphers, a.k.a., public-key encryption schemes.

A number of submissions in response to the call for contributions were received, and are available as ISO document SC 27 N 2656. The author of the present document has carefully reviewed all of the submitted proposals for public-key encryption schemes.

There were a number of different types of schemes submitted:

- Some are based on the hardness of factoring integers or related problems.
- Some are Diffie-Hellman-based schemes — of these, some are based on elliptic curves, and some are based on subgroups of $\mathbf{Z}_p^*$.
- Some allow encryption of arbitrary length messages, and others only allow encryption of short messages.
- Some allow for additional data to be "non-malleably bound" to the ciphertext, while others do not.
- Some allow for messages and ciphertexts to be efficiently processed as "streams," while others do not, requiring more than one pass over this data.
- Some have claims of "provable" security against adaptive chosen ciphertext attack — some relying on the "random oracle" heuristic — some not. For several schemes, these claims of security have proven to be invalid upon closer scrutiny.

Clearly, these submissions are quite incompatible in a number of respects, and one of the challenges of this project is to minimize these incompatibilities.

Recently, an *ad hoc* meeting of the working group was held on August 21 in Santa Barbara, to further discuss the proposals in an earlier draft of this document. The current version of this proposal reflects the author's own vision for how this standard should develop, while at the same time, takes into account the discussions at the *ad hoc* meeting.

## 1.2 Goals of this document

The goals of this document are as follows:

- To propose a standard functionality that a public-key encryption scheme should implement. This is essentially an *abstract interface*.
- To propose a "unified framework" for hybrid encryption. In order for a cryptosystem to be practical, it must be able to process messages that are arbitrary byte strings. There are traditional, and fairly well known "hybrid" schemes to do this: one first uses public-key techniques to derive a shared key, and then encrypts or decrypts the actual payload using symmetric-key techniques. We propose a fairly traditional and "provably" secure way of doing this.

  We shall refer to a method of generating a shared random key in this sense as a "key encapsulation mechanism," and we shall refer to a method of encrypting the message using such a shared random key as a "data encapsulation mechanism."
- To propose a "unified framework" for Diffie-Hellman-based encryption schemes. This framework specifies an "abstract group interface" for a group so that any Diffie-Hellman-based encryption scheme can be specified with respect to an abstract group, but yet the group can be implemented in one of several ways, including as a subgroup of an elliptic curve group, and as a subgroup of $\mathbf{Z}_p^*$. The interface is rich enough so as to support all of the subtleties and quirks found in many proposed cryptosystems, especially those using elliptic curves.
- To propose a set of encryption schemes such that each scheme
  - is "provably" secure against adaptive chosen ciphertext attack in some reasonable sense,
  - conforms to the proposed abstract interface,
  - conforms to the proposed unified framework for hybrid encryption (where applicable),
  - conforms to the proposed unified framework for Diffie-Hellman-based encryption (where applicable),
  - provides a fairly unique and attractive tradeoff between efficiency and security, and
  - conforms to pre-existing standards (where applicable).

In order to achieve the last goal of proposing a set of schemes meeting the stated requirements, we have taken several of the submitted schemes, and proposed modified schemes that meet the stated requirements. Some of these changes are quite minor, while others are more drastic. Some schemes were omitted altogether — given the limited amount of time and other resources available to construct this proposal for a standard, resources had to be concentrated on those schemes which appeared most likely to meet the stated objectives, either with or without minor modification.

## 1.3 Preliminary remarks on security

Typically, practical *symmetric* encryption schemes are designed "from scratch," based partly on established design principles. The security of such a scheme is usually simply taken on faith — there is no justification other than to demonstrate that reasonable design principles were employed in the design of the scheme, and to give (perhaps heuristic) arguments that the scheme resists known types of attacks.

For public-key encryption schemes, the situation is somewhat different. Such a scheme is typically composed of a number of components: besides some kind of "trapdoor" cryptographic transform, there may also be various other components, such as hash functions, symmetric ciphers, etc. Because of this, it is customary nowadays to formally analyze the security of such a scheme relative to the security of its constituent components; that is, to prove the security of the scheme under the assumption that these components satisfy particular, explicit security requirements.

Since proving the security of practical schemes in this way is often infeasible, a heuristic called the *random oracle model* is sometimes used in the proof. In this approach, a cryptographic hash function is modeled — for the purposes of analysis — as a "black box" containing a random function to which the adversary and the algorithms implementing the cryptosystem have "oracle access." This approach has been used implicitly and informally for some time; however, it was formalized by Bellare and Rogaway [BR93], and has subsequently been used quite a bit in the cryptographic research community.

We should stress, however, that the random oracle model is not just "another assumption," like assuming that a hash function is collision resistant, or that a function is pseudo-random. It is a heuristic "leap of faith" — invoking this heuristic is qualitatively a much bigger step than making any particular cryptographic assumption. Indeed, in [CGH98], it is shown that there are cryptosystems that are secure in the random oracle model, but are insecure *no matter what* hash function is used to implement the random oracle.

Despite these problems, the random oracle model is still a useful heuristic and design principle. A proof of security in the random oracle model is still much better than no proof at all, and certainly such a proof does rule out a large family of attacks.

In judging the security of a "provably secure" scheme, there are several independent "dimensions":

- the use or non-use of the random oracle heuristic,
- the "strength" of the underlying assumptions, and
- the efficiency of the security reduction.

Because of these several dimensions, the security of two "provably secure" schemes might be essentially incomparable. For example, one scheme might rely on the random oracle heuristic and a weak assumption, and the other might not use the random oracle heuristic but rely on a stronger assumption, or perhaps the assumptions are simply incomparable.

The efficiency of a security reduction is an issue that is all too often ignored. However, it should be taken into account. For example, a scheme might be secure if RSA inversion is hard, but the security reduction may be so inefficient that for typical sizes of keys — say 1024-bit RSA modulus — the implied algorithm for solving the RSA inversion problem might be slower than the fastest currently known algorithm for factoring numbers.

Even if the security reduction is very inefficient, it can still be argued that such a proof of security nevertheless provides a "qualitative" guarantee of security. Moreover, such a reduction does rule out attacks that would efficiently "scale" to larger sizes of keys.

For public-key encryption schemes, it is widely agreed that the "right" notion of security for a scheme intended for general-purpose use is that of *security against adaptive chosen ciphertext attack.* This notion was introduced in [RS91], and implies other useful properties, like *non-malleability.* See [DDN91, DDN98, BDPR98] for further discussion. In this document, this will be the relevant notion of security used for judging the security of an encryption scheme.

### 1.4 A summary of submissions and proposed schemes

In this section, we summarize the submissions that were made, give a very brief assessment of of the submissions, and briefly describe the schemes that we actually propose to be included in the standard.

#### 1.4.1 *RSA-OAEP*, *RSA-OAEP+*, and *RSA-KEM*

*RSA-OAEP* is the fairly well-established RSA encryption scheme, using the padding scheme OAEP invented by Bellare and Rogaway [BR94], with enhancements and refinements due to Johnson and Matyas [JM96].

The submission coincides with the standards PKCS #1 v2.0 and IEEE P1363.[1]

One of the main supposed virtues of this scheme was an alleged proof in the random oracle model of security against adaptive chosen ciphertext attack, assuming RSA inversion is hard. This "proof" was published in [BR94], and despite years of public scrutiny, it was only recently observed in [Sho01] that not only is the proof invalid, but that there can be *no* standard proof via "black box" reduction for the *OAEP* construction in general, given an *arbitrary* one-way trapdoor permutation.

This negative result does not necessarily imply that the specific instance *RSA-OAEP* is insecure. Indeed, as it turns out — essentially by accident, rather than design — *RSA-OAEP* is indeed secure in the random oracle model. This was proven for encryption exponent 3 in [Sho01], and for arbitrary encryption exponent in [FOPS01]. The security reduction is the latter paper is highly inefficient, however.

Another problem with *RSA-OAEP* is that it only encrypts messages of short length. As such, many applications that use *RSA-OAEP* use it simply as a key encapsulation mechanism, which wastes one of the most attractive features of *RSA-OAEP*, namely, its fairly compact ciphertexts.

To overcome these problems, we propose in this document a new scheme, called *RSA-OAEP+*. It is just as efficient as *RSA-OAEP*, but the general OAEP+ construction is provably secure in the random oracle model (as shown in [Sho01]). Moreover, even with RSA, the security reduction for OAEP+ is much more efficient than that in [FOPS01] for OAEP, making the scheme more attractive from a concrete security point of view. Also, *RSA-OAEP+* is enhanced to deal with arbitrary-length messages in a very "compact" manner.

Even with the security improvements provided by *RSA-OAEP+*, the security reduction is still so inefficient that the security guarantees provided are still not very meaningful in a strict, quantitative sense. For this reason, we also recommend an alternative RSA scheme which is both simpler and more secure, which we call *RSA-KEM*. This scheme is designed as a pure key encapsulation mechanism, and fits more nicely into our framework for hybrid encryption.

#### 1.4.2 *ECIES*

*ECIES* is the "Elliptic Curve Integrated Encryption Scheme." It is a Diffie-Hellman-based scheme. It is a hybrid encryption scheme based on the hardness of the Computational Diffie-Hellman (CDH) problem for elliptic curves. It is closely related to the *DHAES* construction in [ABR98].

The current draft of IEEE P1363a[2] also contains a version of *ECIES*, but this version differs in some significant respects from the submitted version of *ECIES*.

As we shall point out, this scheme is not secure against adaptive chosen ciphertext attack, but can easily be made so with some small changes. Therefore, we have proposed a scheme *ECIES-KEM*,

[1] In this document, all references to IEEE P1363 refer specifically to the final draft D13, dated November 12, 1999.

[2] In this document, all references to IEEE P1363a refer specifically to the latest draft D9.9, dated July 21, 2001.

which besides providing a higher level of security, also has been generalized to conform to our proposed unified frameworks for hybrid and Diffie-Hellman-based encryption. The changes between *ECIES* and *ECIES-KEM* are the minimal changes required to ensure security.

The *ECIES-KEM* scheme is a pure key encapsulation mechanism. It can be converted into a hybrid encryption scheme using the techniques standardized here. The resulting hybrid encryption scheme is consistent with the *ECIES* scheme proposed in IEEE P1363a, in the sense that the scheme proposed here conforms to that in IEEE P1363a; however, there are a number of variations and modes of operation of *ECIES* in P1363a that *do not* conform to the scheme proposed here. Thus, this standard for *ECIES* is a strict subset of the IEEE P1363a standard for *ECIES*.

The main motivation for these restrictions are security concerns about the IEEE P1363a version of *ECIES*. The secondary motivation for these restrictions is to make this standard simpler, more uniform, and self-consistent.

The *ECIES-KEM* scheme can be proven secure against adaptive chosen ciphertext attack, either by using the rather non-standard assumption in [ABR98], or by using the random oracle heuristic, combined with the (also not very standard) assumption that the CDH problem is hard, even when given access to an oracle for the Decisional Diffie-Hellman (DDH) problem. This latter assumption is called the *gap-CDH* assumption, and is studied in detail in [OP01].

As for efficiency, encryption takes two group exponentiations, and decryption takes one or two (depending on the group, but usually one for elliptic curves).

### 1.4.3 *PSEC*

*PSEC* is a family of Diffie-Hellman-based encryption schemes.

It is claimed that these schemes are all provably secure in the random oracle model, under different assumptions. There are three schemes: *PSEC-1*, *PSEC-2*, and *PSEC-3*.

- For *PSEC-1*, the DDH problem is assumed to be hard.
- For *PSEC-2*, the CDH problem is assumed to be hard.
- *PSEC-3* is based on the gap-CDH assumption.

We shall argue that actually, these security claims for *PSEC-1* and *PSEC-2* are unjustified (see §16.5).

We shall propose a scheme *PSEC-KEM* that is a variant of *PSEC-2*, and we provide a complete and detailed proof of security in the random oracle model based on the CDH assumption. Besides correcting the security problems of *PSEC-2*, other changes were made so that the resulting scheme conforms to our proposed abstract interface and to our proposed unified frameworks for hybrid and Diffie-Hellman-based encryption. In particular, as the name implies, *PSEC-KEM* is a key encapsulation mechanism.

As for efficiency, both *PSEC-KEM* encryption and decryption require two group exponentiations.

*PSEC-1* is based on stronger assumptions, is not significantly more efficient than the other schemes, and has some significant security problems. For these reasons, we have chosen not to include it (or a variant thereof) in this proposal.

*PSEC-3* is very similar to *ECIES*, offering an almost identical efficiency/security trade-off; since *ECIES* appears to be the more well-established scheme, we have chosen not to include *PSEC-3* (or a variant thereof) in this proposal.

### 1.4.4 *ACE-Encrypt*

*ACE-Encrypt* is a Diffie-Hellman-based hybrid encryption scheme that can be proven secure against adaptive chosen ciphertext attack assuming the DDH problem is hard. It is the *only* submission that can truly be proven secure — it does not rely on the random oracle heuristic. It is slightly less efficient than *PSEC-2*.

The submission is based on the DDH problem for a subgroup of $\mathbf{Z}_p^*$. We have proposed a variant, *ACE-KEM*. Several changes were made to the original *ACE-Encrypt* scheme so that the resulting scheme conforms to our proposed abstract interface and to our proposed unified frameworks for hybrid and Diffie-Hellman-based encryption. As the name suggests, *ACE-KEM* is a key encapsulation mechanism. *ACE-KEM* and the corresponding hybrid scheme are still provably secure — without the random oracle heuristic — based on the DDH, as well as a couple of other reasonable symmetric-key cryptographic assumptions.

As for efficiency, *ACE-KEM* encryption requires five group exponentiations, and decryption requires either three or four (depending on the group, but usually three for elliptic curves). Several optimizations are available to reduce the effective costs of these exponentiations, however.

We point out that, like *PSEC-KEM*, the scheme *ACE-KEM* can be proven secure in the random oracle model under the weaker CDH assumption, although the security reduction for *ACE-KEM* is much less efficient than that for *PSEC-KEM*. Additionally, it can be shown that *ACE-KEM* is no less secure than *ECIES-KEM*, in the sense that there is a very tight reduction from an attack on *ECIES-KEM* to an attack *ACE-KEM*. That is, any attack on *ACE-KEM* can be converted into an attack on *ECIES-KEM*, where the running time and success probability of the latter attack are essentially the same as for the former attack. This is discussed in detail in §17.6.2. Thus, any fears that the DDH assumption is "too strong" (see [JN01]) can be safely put to rest.

### 1.4.5 *EPOC*

*EPOC* is a family of encryption schemes based on factoring integers of the form $n = p^2q$. There are three variants: *EPOC-1*, *EPOC-2*, and *EPOC-3*.

Security of these schemes is claimed in the random oracle model under one of several assumptions (including the assumption that factoring is hard).

It was the initial judgment of this author that these schemes should not be included in the standard, for the following reasons:

- the theory on which these schemes are based has not been very widely scrutinized, nor have many of the implementation details;
- they do not seem to offer a particularly attractive efficiency/security tradeoff in relation to the other schemes (one drawback in particular is that it is not amenable "stream processing" — see §2.1.3).

However, at the *ad hoc* meeting in Santa Barbara, a consensus was reached that *EPOC-2* should be included in the standard. The reasons given were that

- it is the only proposed scheme whose proof of security is based on factoring (as opposed to RSA inversion);
- the decryption operation may be somewhat faster than for RSA;
- it has been subjected to a certain amount of public scrutiny, and no security problems have yet been discovered;

- it is in IEEE P1363a, and so including it in the ISO standard increases the compatibility of these two standards.

At the present time, we have not yet incorporated a specification *EPOC-2* into this proposal, although the intention is that it will be a part of the first working draft. This specification should be consistent with the IEEE P1363a version of *EPOC-2*.

### 1.4.6 *HIME*

*HIME* is a family of encryption schemes based on factoring integers.

Security of these schemes is claimed in the random oracle model under one of several assumptions (including the assumption that factoring is hard).

It was the judgment of this author that these schemes should not be included in the standard. The main reason for this is that the design of the schemes and the claims of security do not appear to stand on very firm ground. Indeed, many details are missing, and it is not at all clear that these gaps can be filled in. Moreover, none of these schemes have been published anywhere, and therefore have not been widely scrutinized.

Also, it was the general consensus at the *ad hoc* meeting in Santa Barbara that this scheme should not be considered any further.

### 1.4.7 Further references on the submissions

The schemes *RSA-OAEP*, *ECIES*, *PSEC*, *EPOC*, and *ACE-Encrypt* have also been submitted to the Crypto-Nessie evaluation project, and were presented at the first Crypto-Nessie workshop, held in Leuven on November 13-14, 2000.

Besides the ISO document SC 27 N 2656, detailed descriptions of these algorithms are publicly available at `www.cryptonessie.org/workshop`.

### 1.4.8 Summary of proposed schemes

So our proposed schemes are as follows:

- Diffie-Hellman-based schemes:
  - *ECIES-KEM*
  - *PSEC-KEM*
  - *ACE-KEM*
- RSA-based schemes:
  - *RSA-OAEP*
  - *RSA-OAEP+*
  - *RSA-KEM*
- Factoring-based schemes:
  - *EPOC-2*

| Scheme | exponentiations per encryption | exponentiations per decryption | random oracle heuristic | main assumption |
|---|---|---|---|---|
| *ECIES-KEM* | 2 | 1 (or 2) | yes | gap CDH |
| *PSEC-KEM* | 2 | 2 | yes | CDH |
| *ACE-KEM* | 5 | 3 (or 4) | no | DDH |

Table 1: Comparison of Diffie-Hellman-based schemes

The reason for proposing three different RSA-based schemes is that they each offer a unique efficiency/security trade-off. While they are all based on the RSA problem, the security reduction for *RSA-KEM* is much tighter than that for *RSA-OAEP+*, and the security reduction for *RSA-OAEP+* is much tighter than that of *RSA-OAEP*. Additionally, *RSA-KEM* is very simple, and fits more nicely into our general framework for hybrid encryption. The scheme *RSA-OAEP+* is attractive as it yields more compact ciphertexts than the hybrid scheme arising from *RSA-KEM*. The scheme *RSA-OAEP* is included here mainly for compatibility with other standards: it offers no real security or performance benefit compared to the other RSA-based schemes.

The reason for including three different Diffie-Hellman-based schemes is that they seem to each offer a unique efficiency/security trade-off, as summarized in Table 1.

# 2 Public-key encryption and chosen ciphertext attack

## 2.1 Abstract interface

We first define the basic structure of a public-key encryption scheme.

A public-key encryption scheme *PKE* consists of three algorithms:

- A key generation algorithm *PKE.KeyGen*(), that outputs a public key/secret key pair $(PK, SK)$. The structure of *PK* and *SK* depend on the particular scheme.
- An encryption algorithm *PKE.Encrypt*$(PK, L, M, options)$ that takes as input a public key *PK*, a label $L$, a message $M$, and an optional *options* argument, and outputs a ciphertext $C$. Note that $L$, $M$, and $C$ are byte strings. See §2.1.4 below for more on *labels*. See §2.1.6 below for more on the *options* argument.
- A decryption algorithm *PKE.Decrypt*$(SK, L, C)$ that takes as input a secret key *SK*, a label $L$, and a ciphertext $C$, and outputs a message $M$.

In general, the key generation and encryption algorithms will be probabilistic algorithms, while the decryption algorithm is deterministic.

### 2.1.1 Soundness

A basic requirement of any public-key encryption scheme is *soundness*: for any public-key/secret-key pair $(PK, SK)$, for any label/message pair $(L, M)$, any encryption of $M$ with label $L$ under *PK* decrypts with label $L$ under *SK* to the original message $M$. This requirement may be relaxed, so that it holds only for all but an acceptably negligible fraction of public-key/secret-key pairs, and even just for all but an acceptably negligible fraction of encryptions.

### 2.1.2 Message length

It is important to note that messages may be of arbitrary and variable length, although a particular scheme may choose to impose a (very large) upper bound on this length. Thus, our proposed notion of a public-key encryption scheme is essentially a "digital envelope."

Some currently available public-key encryption schemes, like *RSA-OAEP*, only allow for very short messages, and leave it to application engineers to design their own "hybrid" scheme to encrypt long messages (i.e., by encrypting a session key and then using symmetric-key cryptography to encrypt the payload).

However, it seems unrealistic to expect application engineers to correctly design such a secure hybrid scheme. Even PKCS#7 — the standard "digital envelope" mechanism — is not appropriate. The simplest version of this simply encrypts a session key using *RSA-OAEP*, and then encrypts the message using a standard symmetric cipher — no additional integrity checks are made. Because of this, straightforward application of PKCS#7 yields a trivially *malleable* encryption scheme.

Despite all of the above potential problems and limitations, given that some very important existing encryption schemes do impose a small upper bound on the length of a message, we also introduce the notion of a *bounded-length public-key encryption scheme*. Such a scheme *PKE* supports the same interface as that of an ordinary (unbounded) scheme, but only allows messages of length bounded by *PKE.MaxLen*(*PK*).

### 2.1.3 Stream processing

Given that messages may be arbitrarily long, a highly desirable property of any public-key encryption scheme should be that both the encryption and decryption algorithms can be efficiently implemented as *filters*. That is, the message may be presented to the encryption algorithm as an input stream, and the ciphertext should be written to an output stream. The algorithm should never have to rewind these streams, and should be able to process these streams using a small, bounded amount of internal storage, independent of the length of these streams. Similarly, the decryption algorithm should be given access to an input stream representing the ciphertext, and the message should be written to an output stream.

All of the schemes proposed here are amenable to stream processing, with the sole exception of *EPOC-2*.

### 2.1.4 The use of labels

A *label* is a byte string that is effectively bound to the ciphertext in a non-malleable way. It may contain data that is implicit from context and need not be encrypted, but that should nevertheless be bound to the ciphertext. We view a label to be a byte string that is meaningful to the application using the encryption scheme, and that is independent of the implementation of the encryption scheme.

For example, there are key exchange protocols in which one party, say $A$, encrypts a session key $K$ under the public key of the other party, say $B$. In order for the protocol to be secure, party $A$'s identity (or public key or certificate) must be non-malleably bound to the ciphertext. One way to do this is simply to append this identity to the message. However, this creates an unnecessarily large ciphertext, since $A$'s identity is typically already known to $B$ in the context of such a protocol. A good implementation of the labeling mechanism achieves the same effect, without increasing the size of the ciphertext.

Labels may also be of arbitrary and variable length, but we do not impose the restriction that the encryption and decryption algorithms should be able to process labels as streams.

Both the *ECIES* and *RSA-OAEP* submissions include the notion of a label (where it is called an *encoding parameter*), although no indication was given as to the role or function of a label. Nevertheless, it seems to be a potentially useful feature, and so we include it here.

### 2.1.5 Ciphertext format

The schemes proposed in this document describe precisely the format of a ciphertext. This is desirable for several reasons. First, it facilitates the inter-operability of different implementations of the same scheme. Second, it allows higher-level protocols to use public-key encryption as a "black box" in a way that is independent of the particular scheme. Third, it is necessary to even discuss the notion of security in a meaningful way.

It is *highly recommended* that a general-purpose library offering public-key encryption implement the abstract interface in a way suitable for its particular programming environment, and that the ciphertexts conform to the prescribed format. Standardizing the abstract interface and the ciphertext format is meant to facilitate this type of software development.

However, these specifications in no way dictate that such formatting *must* be preserved in a system using an encryption scheme, or that an implementation of the abstract interface *must* explicitly appear anywhere in the system. For example, in transporting a ciphertext over a network, it may be chopped up, and reformatted in an arbitrary way. Some transformations may be generic, i.e., independent of the encryption scheme, while others may be applicable only to a specific scheme. Indeed, it is not even necessary that a particular system using a scheme standardized here actually outputs (resp., inputs) ciphertexts in the prescribed format upon encryption (resp., decryption): the encryption and decryption algorithms may behave *as if* they performed such transformations on the ciphertext, even though the ciphertext may never really be represented in the prescribed format. In such a system, the prescribed format of the ciphertext plays a purely conceptual role in reasoning about the security of the system, even though it plays no direct role in the implementation of the system.

### 2.1.6 Scheme-specific encryption options

Some schemes allow certain types of scheme-specific options to be passed to the encryption algorithm, which is why we have allowed for an extra argument *options* in our abstract interface.

Allowing scheme-specific options in an abstract interface is clearly not such a good idea, as this runs counter to the very notion of an abstract interface. Also, since such options are scheme specific, their use will almost certainly atrophy over time, especially if more applications take advantage of the benefits provided by an abstract encryption interface.

Although we strongly discourage the use of such options, we nevertheless allow them in our formal model for the sake of allowing us to cast some existing schemes into the model.

The only place where we will actually use encryption options is in discussing encryption schemes based on elliptic curves: some of these scheme allows the encryptor to *dynamically* choose, from one of several formats, how it wants to format a point on the curve.

For the sake of some notion of completeness or symmetry, one could also allow scheme-specific decryption options, but since we do not need such a notion, we do not attempt to formalize it.

### 2.1.7 Algorithm failure

Throughout this document, algorithms will always compute a function on their inputs, except that instead of returning a value, they may **fail**. By convention, if an algorithm **fails**, then unless otherwise specified, an algorithm that invokes that algorithm as a sub-routine also **fails**. Thus, **failing** is analogous to "throwing an exception" in many programming languages.

### 2.1.8 Bits vs. Bytes

Our abstract interface treats messages and labels as strings of *bytes* (a.k.a., *octets*), rather than strings of *bits*.

We argue against allowing bit strings, for the following reasons.

First, it seems unlikely that many, if indeed any, applications really work with data that is represented as bit strings, rather than byte strings.

Second, even if an application does want to work with bit strings, it can be left to the application to encode these bit strings as byte strings. Anyway, an application may in general have to properly format its messages before encryption, for example, to hide any information that may be leaked just from the length of the message (see §2.2.2).

Third, we note that existing standards, such as IEEE P1363a, are themselves inconsistent in this regard, and so it does not seem advisable to propagate this inconsistency here as well. Indeed, while some encryption schemes in IEEE P1363a (*ECIES*) allow for bit strings, others (*RSA-OAEP* and *EPOC-2*) are byte oriented.

## 2.2 Notion of security

We next recall the definition of security against adaptive chosen ciphertext attack, adapted to deal with labels.

We begin by describing the attack scenario.

First, the key generation algorithm is run, generating the public key and private key for the cryptosystem. The adversary, of course, obtains the public key, but not the private key.

Second, the adversary makes a series of arbitrary queries to a *decryption oracle*. Each query is a label/ciphertext pair $(L, C)$ that is decrypted by the decryption oracle, making use of the private key of the cryptosystem. The resulting decryption is given to the adversary; moreover, if the decryption algorithm **fails**, then this information is given to the adversary, and the attack continues. The adversary is free to construct these label/ciphertext pairs in an arbitrary way—it is certainly *not* required to compute them using the encryption algorithm.

Third, the adversary prepares a label $L^*$ and two "target" messages $M_0, M_1$ of equal length, and gives these to an *encryption oracle*. If the scheme supports any encryption options, the adversary also chooses these. The encryption oracle chooses $b \in \{0, 1\}$ at random, encrypts $M_b$ with label $L^*$, and gives the resulting "target" ciphertext $C^*$ to the adversary.

Fourth, the adversary continues to submit label/ciphertext pairs $(L, C)$ to the decryption oracle, subject only to the restriction that $(L, C) \neq (L^*, C^*)$.

Just before the adversary terminates, it outputs $\hat{b} \in \{0, 1\}$.

That completes the description of the attack scenario.

For a given adversary $A$, and a given scheme *PKE*, we define the adversary's *guessing advantage*

$$Advantage_{PKE}(A) = \left|\Pr[\hat{b} = b] - 1/2\right|.$$

Security means that $Advantage_{PKE}(A)$ is "acceptably" small for all adversaries $A$ that run in a "reasonable" amount of time.

### 2.2.1 Alternative characterizations and implications

Note that in proving an encryption scheme secure, the definition we have given is usually the most convenient. However, in analyzing an encryption scheme in a larger context, a slightly different definition is usually more convenient. In this definition, the attack scenario proceeds just as before. However, instead of measuring the adversary's guessing advantage, we measure its *distinguishing advantage*

$$Advantage'_{PKE}(A) = \left|\Pr\left[\hat{b} = 1 | b = 1\right] - \Pr\left[\hat{b} = 1 | b = 0\right]\right|.$$

It follows directly from the definitions by a trivial calculation that for any adversary $A$,

$$Advantage'_{PKE}(A) = 2 \cdot Advantage_{PKE}(A).$$

Equivalently, one can view $Advantage'_{PKE}(A)$ as measuring how differently an adversary behaves in two different attack games: in one game, $M_0$ is always encrypted, and in the other, $M_1$ is always encrypted.

In analyzing an encryption scheme in a larger context, one usually substitutes an encryption of a secret message by an encryption of a garbage message (all zeros, or random) of the same length, and then analyzes how the adversary behaves. Many secret messages may be replaced by garbage strings, and the distinguishing advantage simply sums (although for some schemes, one can exhibit an even tighter security reduction). A small distinguishing advantage implies that the adversary will not behave significantly differently when this substitution is made. See [BBM00] for more details.

This definition, in slightly different form, was first proposed by Rackoff and Simon [RS91]. It is generally agreed in the cryptographic research community that this is the "right" security property for a general-purpose public-key encryption scheme. This notion of security implies other useful properties, like *non-malleability*. See [DDN91, DDN98, BDPR98] for more on notions of security for public-key encryption schemes.

### 2.2.2 Hiding the message length

Note that in the attack game, the adversary is required to submit two target messages of *equal* length to the encryption oracle. This restriction on the adversary reflects the fact that we cannot expect to hide the length of an encrypted message from the adversary—for many cryptosystems, this will be evident from the length of the ciphertext. It is in general up to the *application* using the cryptosystem to ensure that the length of a message does not reveal sensitive information.

For bounded-length public-key encryption schemes (see §2.1.2) the notion of security is the same as for the ordinary, unbounded case, except that the adversary *is not* required to submit target messages of equal length to the encryption oracle. This reflects the fact that such schemes in fact do hide the length of an encrypted message from the adversary.

### 2.2.3 Failing streams

There is a subtle interaction between **failing**, as discussed in §2.1.7 and the notion of a stream, discussed in §2.1.3. An application reading the output stream of the decryption algorithm should take care not to leak any information about the message it has read from that stream, until the decryption process has finished without **failing**. If it does not do this, the application could potentially forfeit the security guarantees of the scheme.

### 2.2.4 Security parameters and asymptotic security

Note that none of these definitions make explicit mention of a *security parameter*. Our point of view is concrete—not asymptotic. We assume that a scheme specifies a *particular* security parameter (or set of parameters). If one wants to translate these definitions into ones compatible with the "asymptotic complexity" point of view, then one can consider families of algorithms indexed by a parameter $\lambda = 1, 2, 3, \ldots$ that run in time bounded by a polynomial in $\lambda$. Both the scheme and the adversary are viewed as families of algorithms. One can consider either uniform or non-uniform families of algorithms. Security means that the adversary's advantage is "negligible" in $\lambda$, meaning that it goes to zero faster than the inverse of any polynomial in $\lambda$.

## 2.3 Benign malleability: a slightly weaker notion of security

The definition of security given in §2.2 is sometimes viewed as being unnecessarily strong. For example, suppose we take an encryption scheme $PKE$ that satisfies the definition in §2.2, and modify it as follows, obtaining a new encryption scheme $PKE'$: the scheme $PKE'$ is the same as $PKE$, except that it appends a random byte to the ciphertext upon encryption, and ignores this extra byte upon decryption. Technically speaking, $PKE'$ does not satisfy the definition in §2.2, yet this seems counter-intuitive. Indeed, although $PKE'$ is technically "malleable," it is only malleable in a "benign" sort of way: one can create alternative encryptions of the same message, and these alternative encryptions are all clearly recognizable as such.

We present here a formal notion of security that precisely captures the intuitive notion of "benign malleability."

For a particular encryption scheme $PKE$ and any public-key/secret-key pair $(PK, SK)$, we call a binary relation $\equiv_{PK}$ on label/ciphertext pairs a *compatible* relation if for any and two label/ciphertext pairs $(L, C)$ and $(L', C')$,

- if $(C, L) \equiv_{PK} (C', L')$ then $L = L'$,
- if $(C, L) \equiv_{PK} (C', L')$ then $PKE.Decrypt(SK, L, C) = PKE.Decrypt(SK, L', C')$, and
- it can be determined if $(C, L) \equiv_{PK} (C', L')$ using an efficient algorithm that takes inputs $PK, L, C, L', C'$.

Clearly, any compatible relation is an equivalence relation.

We say that a public-key encryption scheme $PKE$ is *benignly malleable* if there exists a compatible relation for $PKE$ as above, and if it satisfies the definition of security given in §2.2, but with the following modification in the attack game: when the adversary submits a label/ciphertext pair $(L, C)$ to the decryption oracle subsequent to the invocation of the encryption oracle, then instead of requiring that $(L, C) \neq (L^*, C^*)$, we only require that $(L, C) \not\equiv_{PK} (L^*, C^*)$.

This definition of security is essentially just as useful as the definition in §2.2 for all applications of public-key encryption that we know of.

# 3 Key encapsulation

In designing an efficient public-key encryption scheme, a useful approach is to design a "hybrid" scheme, where one uses public key cryptography to encrypt a key that can then be used to encrypt the actual payload using symmetric key cryptography.

We will build a hybrid encryption scheme from two lower-level "building blocks." The first is a method for using public key cryptography to "encapsulate" a symmetric key. We call such a scheme a *key encapsulation mechanism*. The second is a method to properly encrypt the message using symmetric-key cryptographic techniques applied to the the symmetric key obtained from the key encapsulation mechanism. We call such a scheme a *data encapsulation method*.

In this section, we discuss key encapsulation mechanisms. Data encapsulation mechanisms are discussed in §4, and the construction of hybrid encryption schemes built out of key and data encapsulation mechanisms is discussed in §5.

Briefly, a key encapsulation mechanism works just like a public-key encryption scheme, except that the encryption algorithm takes no input other than the recipient's public key. Instead, the encryption algorithm generates a pair $(K, C_0)$, where $K$ is a bit string of some specified length, and $C_0$ is an encryption of $K$, that is, the decryption algorithm applied to $C_0$ yields $K$.

One can always use a (possibly bounded-length) public-key encryption scheme for this purpose, generating a random bit string, and then encrypting it under the recipient's public key. However, as we shall see, one can construct a key encapsulation scheme in other, more efficient, ways as well.

Now the details.

## 3.1 Abstract interface

A key encapsulation mechanism *KEM* consists of three algorithms.

- A key generation algorithm *KEM.KeyGen*(), that outputs a public key/secret key pair $(PK, SK)$. The structure of $PK$ and $SK$ depend on the particular scheme.
- An encryption algorithm *KEM.Encrypt*$(PK, options)$ that takes as input a public key $PK$, along with an optional *options* argument, and outputs a key/ciphertext pair $(K, C_0)$. The role of *options* is analogous to that for public-key encryption (see §2.1.6).
- A decryption algorithm *KEM.Decrypt*$(SK, C_0)$ that takes as input a secret key $SK$ and a ciphertext $C_0$, and outputs a key $K$.

A key encapsulation mechanism also specifies a positive integer *KEM.OutputKeyLen* — the length of the key output by *KEM.Encrypt* and *KEM.Decrypt*.

Any key encapsulation mechanism should satisfy a soundness property analogous to the soundness property of a public-key encryption scheme, as described in §2.1.1.

Additionally, we need the following property. The set of all possible outputs of the encryption algorithm should be a subset of some an easy-to-recognize, prefix-free language.[3] The prefix-freeness property is needed so that we can parse byte strings from left to right, and efficiently "strip off" a ciphertext. Note that if all ciphertexts have the same length, then the prefix-freeness property is trivially satisfied.

## 3.2 Notion of security

We next define security against adaptive chosen ciphertext attack for a key encapsulation mechanism.

We begin by describing the attack scenario.

[3]A language $L$ is *prefix free* if for any two $x, y \in L$, $x$ is not a proper prefix of $y$.

First, the key generation algorithm is run, generating the public key and private key for the cryptosystem. The adversary, of course, obtains the public key, but not the private key.

Second, the adversary makes a series of arbitrary queries to a *decryption oracle*. Each query is a ciphertext $C_0$ that is decrypted by the decryption oracle, making use of the private key of the cryptosystem. The resulting decryption is given to the adversary; moreover, if the decryption algorithm **fails**, then this information is given to the adversary, and the attack continues.

Third, the adversary invokes an *encryption oracle*, supplying an encryption option, if the scheme supports them. The encryption oracle does the following:

1. Run the encryption algorithm, generating a pair $(K^*, C_0^*)$.

2. Generate a random string $\tilde{K}$ of length $KEM.OutputKeyLen$.

3. Choose $b \in \{0,1\}$ at random.

4. If $b = 0$, output $(K^*, C_0^*)$; otherwise output $(\tilde{K}, C_0^*)$.

Fourth, the adversary continues to submit ciphertexts $C_0$ to the decryption oracle, subject only to the restriction that $C_0 \neq C_0^*$.

Just before the adversary terminates, it outputs $\hat{b} \in \{0,1\}$.

That completes the description of the attack scenario.

For an adversary $A$, the quantities $Advantage_{KEM}(A)$ and $Advantage'_{KEM}(A)$ are defined in exactly the same way (in terms of $b$ and $\hat{b}$) as $Advantage_{PKE}(A)$ and $Advantage'_{PKE}(A)$ for a public-key encryption scheme (see §2). Security means that $Advantage_{KEM}(A)$ is "acceptably" small for all adversaries $A$ that run in a "reasonable" amount of time.

### 3.3 Further remarks

One can also define the notion of *benign malleability* for key encapsulation mechanisms, just as we did for public-key encryption schemes (§2.3).

Although one could do so, we have chosen not to incorporate the notion of a *label* in the definition of a key encapsulation mechanism. The reason is that the only application we have of a key encapsulation mechanism in this paper is in the construction of a hybrid encryption scheme, and it is easier to implement labels in the data encapsulation mechanism than in the key encapsulation mechanism itself.

## 4 Data encapsulation

A *data encapsulation mechanism* provides a "digital envelope" that protects the secrecy and integrity of data using symmetric-key cryptographic techniques; it also may bind the data to a public label. The definition of security for this primitive that we present here is appropriate for building hybrid public-key encryption schemes, but may not be appropriate for other applications.

In this section we only describe the abstract interface and security properties that a data encapsulation mechanism should satisfy. Later, in §10, we present a fairly traditional and provably secure implementation of a data encapsulation mechanism. There may, however, be other interesting ways to implement this primitive.

### 4.1 Abstract interface

A data encapsulation mechanism *DEM* specifies a key length *DEM.KeyLen*, along with encryption and decryption algorithms:

- The encryption algorithm *DEM.Encrypt*$(K, L, M)$ takes as input a key $K$, a label $L$, and and a message $M$. It outputs a ciphertext $C_1$. Here, $K$, $L$, $M$, and $C_1$ are byte strings, and $L$ and $M$ may have arbitrary length, and $K$ is of length *DEM.KeyLen*.

- The decryption algorithm *DEM.Decrypt*$(K, L, C_1)$ takes as input a key $K$, a label $L$, and a ciphertext $C_1$. It outputs a message $M$.

We assume that the encryption and decryption algorithms are deterministic, and that the scheme is (perfectly) sound, in the sense that for all keys $K$, all labels $L$, and all messages $M$,

$$\mathit{DEM.Decrypt}(K, L, \mathit{DEM.Encrypt}(K, L, M)) = M.$$

### 4.2 Notion of security

We shall need the following security property.

Consider the following attack scenario. The adversary generates two messages (byte strings) $M_0, M_1$ of equal length, and a label $L^*$. A random key $K$ is generated. A random bit $b$ is chosen, and $M_b$ is encrypted under key $K$. The resulting ciphertext $C_1^*$ is given to the adversary. The adversary then submits a series of requests to a *decryption oracle*: each such request is a label/ciphertext pair $(L, C_1) \neq (L^*, C_1^*)$, and the decryption oracle responds with the decryption of $C_1$ with label $L$ under key $K$. The adversary makes a guess $\hat{b}$ at $b$. The adversary's advantage is defined as $|Pr[\hat{b} = b] - 1/2|$. We denote this advantage by $\mathit{Advantage}_{\mathit{DEM}}(A)$.

Security means that this advantage is acceptably small.

## 5 Hybrid encryption

We now propose a canonical way to compose a secure key encapsulation mechanism and a secure data encapsulation mechanism so as to obtain a secure public-key encryption scheme.

Let *KEM* be a key encapsulation mechanism and let *DEM* be a data encapsulation mechanism. To compose these two mechanisms, we require that they are *compatible*, in the sense that *KEM.OutputKeyLen* = *DEM.KeyLen*. So let us assume that *KEM* and *DEM* are compatible in this sense.

We now define a *hybrid* public-key encryption scheme $\mathit{H\text{-}PKE} = \mathit{H\text{-}PKE}_{\mathit{KEM},\mathit{DEM}}$ in terms of *KEM* and *DEM* as follows.

The key generation algorithm, public key, and secret key for *H-PKE* are the same as that of *KEM*.

Let $(PK, SK)$ be a public key/secret key pair.

To encrypt a message $M$ with label $L$ and any encryption *options* under $PK$, the encryption algorithm for *H-PKE* does the following. First, it runs the encryption algorithm for *KEM* with the given *options*, generating a ciphertext $C_0$ and a key $K$. Second, it encrypts $M$ with label $L$ under $K$ using the encryption algorithm for *DEM*. Third, it outputs the ciphertext $C = C_0 \parallel C_1$.

To decrypt a ciphertext $C$ with label $L$ under *SK*, the decryption algorithm for *H-PKE* does the following. First, it parses $C$ as $C = C_0 \parallel C_1$, using the prefix-freeness property of the ciphertexts associated with *KEM*. Second, it decrypts $C_0$ under *SK* using the decryption algorithm of *KEM* to obtain a key $K$. Third, it decrypts $C_1$ with label $L$ under $K$ using the decryption algorithm of *DEM*, and outputs the resulting message $M$. Any of these steps may fail, in which case the decryption algorithm for *H-PKE* also fails.

It is an easy matter to prove that for any adversary $A$,

$$\begin{aligned} Advantage_{H\text{-}PKE}(A) &\leq Advantage'_{KEM}(A_1) + \\ &\quad Advantage_{DEM}(A_2), \end{aligned}$$

where $A_1$ and $A_2$ are adversaries that run in time roughly the same as that of $A$. Actually, this estimate assumes that *KEM* is *perfectly* sound; however, if the soundness condition may be violated with some small probability, that probability must be added into this estimate as well.

It follows that if *KEM* and *DEM* are secure, then so is *H-PKE*.

# 6 Byte string/integer conversions

We simply adopt the functions *OS2IP* and *I2OSP* from the IEEE P1363 standard for conversions between byte (a.k.a., octet) strings and integers.

The function $OS2IP(x)$ takes as input a byte string, and outputs an integer. If $x = x_{l-1} \,\|\, x_{l-2} \,\|\, \cdots \,\|\, x_0$, where each $x_i$ is a byte, then

$$OS2IP(x) = \sum_{i=0}^{l-1} x_i \cdot 256^i.$$

In this formula, each byte $x_i$ is interpreted as a base-256 digit. Note that the left-most byte represents the most-significant digit.

The function *I2OSP* is essentially the inverse of *OS2IP*. The function $I2OSP(m, l)$ takes as input two non-negative integers $m$ and $l$, and outputs the unique byte string $x$ of length $l$ such that $OS2IP(x) = m$, if such an $x$ exists. Otherwise, the function **fails**. Note that the function fails if and only if $m \geq 256^l$.

# 7 Pseudo-random byte generator

A pseudo-random byte generator *PRBG* is a scheme with the following interface. It defines fixed seed length *PRBG.SeedLen* and a function $PRBG.eval(s, l)$ that takes as input a byte string $s$ of length *PRBG.SeedLen* and an integer $l \geq 0$, and produces as output a byte string of length $l$.

The assumption we make is that for a random seed $s$, the output is computationally indistinguishable from a random byte string of the same length.

One recommended way to implement a *PRBG* is to simply use a block cipher in counter mode.

An alternative is to use a block cipher in counter mode, but to output the XOR of consecutive pairs of block cipher outputs. This approach yields a higher level of security when $l$ is very large (see [Luc00]).

# 8 Symmetric key encryption

## 8.1 Abstract interface

A symmetric key encryption scheme *SKE* specifies a key length *SKE.KeyLen*, along with encryption and decryption algorithms:

- The encryption algorithm $SKE.Encrypt(k, M)$ takes as input a key (byte string) $k$ of length $SKE.KeyLen$ and a message $M$. It outputs a ciphertext $c$.
- The decryption algorithm $SKE.Decrypt(k, c)$ takes as input a key $k$ of length $SKE.KeyLen$ and a ciphertext $c$. It outputs a message $M$.

We assume that the encryption and decryption algorithms are deterministic, and that the scheme is (perfectly) sound, in the sense that for all keys $K$ and all messages $M$,

$$SKE.Decrypt(K, SKE.Encrypt(K, M)) = M.$$

### 8.2 Notion of security

We shall need the following security property.

Consider the following attack scenario. The adversary generates two messages (byte strings) $M_0, M_1$ of equal length. A random key $k$ is generated. A random bit $b$ is chosen, and $M_b$ is encrypted under key $k$. The resulting ciphertext $c$ is given to the adversary. The adversary makes a guess $\hat{b}$ at $b$. The adversary's advantage is defined as $|Pr[\hat{b} = b] - 1/2|$.

For an adversary $A$ that chooses $M_0, M_1$ of length bounded by $l$, we denote this advantage by $Advantage_{SKE}(A, l)$.

Security means that the advantage is acceptably small.

Note that one can build a secure symmetric key encryption scheme by using a pseudo-random byte generator (see §7) to generate a "one time pad," which is then XORed with the message.

Also, the IEEE P1363a standard specifies other methods based on block ciphers that could be used as well. The ISO working group should perhaps consider other methods as well.

## 9 One-time MAC

A one-time message authentication code $MAC$ is a scheme that defines two quantities $MAC.KeyLen$ and $MAC.TagLen$, along with a function $MAC.eval(k', T)$ that takes a key $k'$ of length $MAC.KeyLen$ and an arbitrary byte string $T$ as input, and computes as output a byte string $tag$ of length $MAC.TagLen$. We shall need the following security property.

Consider the following attack scenario. A byte string $T^*$ is chosen by the adversary. A key $k'$ is chosen at random. The $MAC$ is evaluated at $T^*$ with key $k'$, and the output $tag^*$ is given to the adversary. The adversary outputs a list of pairs $(T, tag)$, where $T$ is a byte string with $T \neq T^*$ (and not necessarily of the same length as $T^*$), and $tag$ is a byte string of length $MAC.TagLen$. The adversary's advantage is defined to be the probability that for one such pair $(T, tag)$, the $MAC$ on input $T$ with key $k'$ is equal to $tag$.

For an adversary $A$ that chooses $T^*$ of length bounded by $l_1$ and at most $l_3$ pairs $(T, tag)$ with $T$ of length bounded by $l_2$, we denote this advantage by $Advantage_{MAC}(A, l_1, l_2, l_3)$.

Security means that this advantage should be acceptably small.

There are a number of acceptable one-time MAC schemes.

Any of the schemes specified by the ISO MAC standard should be acceptable. IEEE P1363a standard also specifies a MAC that should be acceptable.

# 10 *DEM1*

Given a symmetric encryption scheme *SKE* (see §8), and a one-time message authentication code *MAC* (see §9), here is how one can build a data encapsulation mechanism $DEM1 = DEM1_{SKE,MAC}$. We require that $DEM1.OutputKeyLen = SKE.KeyLen + MAC.KeyLen$.

To encrypt a message $M$ with label $L$ under a key $K$, we parse $K$ as $K = k \,\|\, k'$, where $|k| = SKE.KeyLen$ and $|k'| = MAC.KeyLen$. We encrypt $M$ using *SKE* under key $k$, obtaining its encryption $c$. Then we apply *MAC* to the byte string string $T = c \,\|\, L \,\|\, I2OSP(8 \cdot |L|, 8)$ using $k'$, obtaining *tag*. The entire ciphertext is $C_1 = c \,\|\, tag$.

To decrypt a a ciphertext $C_1$ with respect to a given label $L$, we first parse $C_1$ as $c \,\|\, tag$, where $|tag| = MAC.TagLen$. This step may fail, of course, if $C_1$ is too short. We then parse $K$ as $K = k \,\|\, k'$, where $|k| = SKE.KeyLen$ and $|k'| = MAC.KeyLen$. Then we apply the MAC with key $k'$ to the byte string string $T = c \,\|\, L \,\|\, I2OSP(8 \cdot |L|, 8)$, and test whether the resulting tag equals the given *tag*. If not, we report failure. Otherwise, decrypt $c$ under $k$, obtaining $M$. It is possible that *SKE.Decrypt* fails. Finally, we output $M$.

It is straightforward to show that if the underlying components are secure, then the resulting data encapsulation mechanism *DEM1* is secure. Moreover, the reduction is quite "tight" quantitatively.

Specifically, we have the following:

$$\begin{aligned} Advantage_{DEM1}(A) \;\; \le \;\; & Advantage_{SKE}(A_1, l_1) \; + \\ & Advantage_{MAC}(A_2, l_2, l_3, q_D). \end{aligned}$$

Here,

- $A_1, A_2$ are adversaries that run in about the same time as $A$,
- $l_1$ is a bound on the length of the target message,
- $l_2$ is a bound on the length of the string $T^*$ corresponding to the target ciphertext,
- $l_3$ is a bound on the length of the strings $T$ corresponding to ciphertexts submitted to the decryption oracle,
- $q_D$ is a bound on the number of decryption oracle queries,
- $Advantage_{SKE}$ is as defined in §8, and
- $Advantage_{MAC}$ is as defined in §9.

The proof of this is an easy exercise.

Note that we pass an encoding of the length of $L$ to *MAC*. This is essential to ensure non-malleability (see §15.6.3). The particular mechanism used to encode the length was chosen so as to be compatible with the IEEE P1363a version of *ECIES*. In particular, we multiply $|L|$ by 8 so as to encode the length of $L$ in *bits*, rather than *bytes*, since the IEEE P1363a version of *ECIES* allows (in theory, but not really in practice) messages and labels that are bit strings rather than byte strings.

We continue here the discussion started in §2.2.3. In our hybrid construction, there is a single *tag* that is checked at the end of the ciphertext stream. This is the simplest approach, and one that is already seen in practice (as in *ECIES*). In the *ACE-Encrypt* submission, there was actually a *tag* value inserted every kilobyte or so in the ciphertext stream. The reason for this was so that the decryption algorithm would **fail** as soon as it detected a "bad" ciphertext stream. This would greatly enhance the ability of an application to process the output stream of the decryption algorithm in a stream-like fashion — it would not have to wait until the end of the output stream to detect a "bad" stream. It would not be too difficult to specify such a scheme. This point should perhaps be discussed by the working group.

## 11 Hash functions

We shall assume the availability of a cryptographic hash function. Let *Hash* denote the scheme. Then *Hash.OutputLen* denotes the length of the hash function output, and *Hash.eval* denotes the hash function itself, which maps arbitrary length byte strings to byte strings of length *Hash.OutputLen*.

The invocation of *Hash.eval* may fail if the input length exceeds some (very large) implementation-defined bound.

In the security analysis, we shall make the following types of assumptions about *Hash*:

- It is *collision resistant*, i.e., it is hard to find two inputs $x, y$ with $x \neq y$ such that $\mathit{Hash.eval}(x) = \mathit{Hash.eval}(y)$.

- It is *second-preimage collision resistant*, i.e., for a given set $S$ of byte strings together with a prescribed probability distribution on $S$, if $x \in S$ is chosen at random, then it is hard to find $y \in S$ with $x \neq y$ such that $\mathit{Hash.eval}(x) = \mathit{Hash.eval}(y)$. The set $S$ and the probability distribution depend on the application.

- It is a good *entropy-smoothing hash function*, i.e., for a given set $S$ of byte strings together with a prescribed probability distribution on $S$, then if $x \in S$ is chosen at random, the output $\mathit{Hash.eval}(x)$ is computationally indistinguishable from a random byte string of length *Hash.OutputLen*. Of course, for this assumption to be reasonable, it must be the case that the *entropy* of $S$ is sufficiently high.

- We might also choose to view it as a *random oracle*.

Recommended choices for *Hash* are SHA-1 and RIPEMD-160.

## 12 Key derivation functions

It is convenient to have a key derivation function $\mathit{KDF}(x, l)$ that takes as input a byte string $x$ and an integer $l \geq 0$, and outputs a byte string of length $l$. The string $x$ is of arbitrary length.

The invocation of *KDF* may fail if the input or output lengths exceed some (very large) implementation-defined bound.

In the security analysis, we will often model *KDF* as a random oracle.

A specific security property that is sometimes desirable for a key derivation function is that it be a good *entropy smoothing function*. That is, the input $x$ is chosen at random from a distribution of byte strings with high entropy, then the output should be computationally indistinguishable from a random byte string of the same length.

Sometimes the notion of a key derivation function is called a *mask generating function (MGF)*; there seems to be no difference in meaning between the two terms.

### 12.1 *KDF1*

This function is parameterized by a hash function *Hash* (see 11), and is defined as follows. On input $(x, l)$, the output is the first $l$ bytes of

$$Hash.eval(x \,\|\, I2OSP(0,4)) \,\|\, \cdots \,\|\, Hash.eval(x \,\|\, I2OSP(k-1,4)),$$

where $k = \lceil l / Hash.OutputLen \rceil$.

This function is the same as the function called *MGF1* in IEEE P1363.

### 12.2 *KDF2*

This function *KDF2* is the same as *KDF1*, except that the counter runs from 1 to $k$, rather than from 0 to $k-1$.

This function is the same as the function called *KDF2* in IEEE P1363a, except that the latter allows for an output that is a *bit* string, rather than a *byte* string, and also allows for an extra *key derivation parameter* that we do not need here.

### 12.3 Security critique of *KDF1* and *KDF2*

Of course, if one chooses to model *KDF1* (or *KDF2*) as a random oracle in a security analysis, one is free to do so. There is really not much of a rational basis to argue either for or against such a choice.

However, we do not recommend the use of these functions in applications where one requires the entropy smoothing property discussed above. The only point in this document where this is significant is in the analysis of the variant of the *ACE-Encrypt* scheme discussed in §17, whose security analysis is not based on the random oracle heuristic.

Our reasoning is as follows.

If we were to believe that these were good entropy smoothing functions, this would suggest that the function $F_x(y)$ defined by

$$F_x(y) = Hash.eval(x \,\|\, y)$$

should be a "good" pseudo-random function with key $x$ and input $y$. However, standard hash functions, like SHA-1, are built using a particular block cipher $P_a(b)$ — with key $a$ and input block $b$ — chained in a standard way. Indeed, suppose that *Hash* is SHA-1 with initial chaining value $IV$ and that $x$ is 512-bits long. So in this case, $P_a(b)$ is a block cipher with a 512-bit key size, and a 160-bit block size. Then

$$F_x(y) = P_y(z) \oplus z, \text{ where, } z = P_x(IV) \oplus IV.$$

Assuming that $P_a(b)$ is a good block cipher, and that $x$ is suitably random, then the value $z$ above should be pseudo-random. Therefore, the security of $F_x(y)$ as a pseudo-random function is equivalent to the security of the function $G_z(y)$ defined by

$$G_z(y) = P_y(z) \oplus z$$

as a pseudo-random function with key $z$ and input $y$. Is $G_z(y)$ a good pseudo-random function? This is not clear. But certainly, this is a quite unorthodox construction that does not appear to be based on any well-worn or otherwise sound principles.

Because of this perceived potential weakness, we propose two further key derivation functions, *KDF3* and *KDF4*. Either of these can be used in a situation where a random oracle is required. However, these functions seem more reasonable in applications where the entropy smoothing property is required.

### 12.4 *KDF3*

This function is parameterized by a hash function *Hash* and a padding amount $pamt \geq 4$, and is defined as follows. On input $(x, l)$, the output is the first $l$ bytes of

$$Hash.eval(I2OSP(0, pamt) \,\|\, x) \,\|\, \cdots \,\|\, Hash.eval(I2OSP(k-1, pamt) \,\|\, x),$$

where $k = \lceil l / Hash.OutputLen \rceil$.

Recommended choices for the hash function are SHA-1 or RIPEMD-160. Recommended choices for *pamt* are either 4, or the block size of the underlying hash (64 in the case of SHA-1 or RIPEMD-160).

Based upon the way standard hash functions like SHA-1 or RIPEMD-160 are constructed, it seems like a reasonable assumption is that they are good pseudo-random functions, where we view the text input as the key of the function, and we view the initial vector $IV$ as the input to the function. Typical implementations of these hash functions often do not provide an interface that allows the programmer to choose the $IV$. However, we get the equivalent effect by setting *pamt* to the block size of the underlying hash.

By setting *pamt* to the block size of the underlying hash function, we are able to give a reasonable justification for the security of *KDF3*. If we set *pamt* to another value, such as 4, this justification is no longer valid. Nevertheless, setting $pamt = 4$ does not seem like a completely unreasonable choice, and certainly the arguments we made above against *KDF1* and *KDF2* no longer apply.

### 12.5 *KDF4*

This function is parameterized by a hash function *Hash* and a pseudo-random byte generator *PRBG* (see §7). It is required that $Hash.OutputLen = PRBG.SeedLen$.

On input $(x, l)$, this function outputs

$$PRBG.eval(Hash.eval(x), l).$$

For the hash function, one can use a standard function like SHA-1 or RIPEMD-160. If *PRBG.SeedLen* is less than 20, then one can simply truncate the output of the hash function.

This function will be a good entropy smoothing function, provided *Hash* is a good entropy smoothing function, and provided *PRBG* is secure as a pseudo-random byte generator.

## 13 Abstract groups

We describe a group as an abstract data type. As a matter of convention, we shall always use additive notation for a group. Also, group elements will be typeset in boldface, and $\mathbf{0}$ denotes the identity element of the group.

A *fully specified* group *Group* is a tuple $(\mathcal{H}, \mathcal{G}, \mathbf{g}, \mu, \nu, \mathcal{E}, \mathcal{D}, \mathcal{E}', \mathcal{D}')$, where:

- $\mathcal{H}$ is a finite abelian group in which all group computations are actually performed. Note that this group need not be cyclic.
- $\mathcal{G}$ is a *cyclic* subgroup of $\mathcal{H}$. This is where the real "action" will normally take place in a cryptographic scheme.
- $\mathbf{g}$ is a generator for $\mathcal{G}$.
- $\mu$ is the order (size) of $\mathcal{G}$, and $\nu$ is the index of $\mathcal{G}$ in $\mathcal{H}$, i.e., $\nu = |\mathcal{H}|/\mu$.

  We shall require that $\mu$ is prime. For some cryptographic schemes, we make the stronger requirement that $\gcd(\mu, \nu) = 1$.
- $\mathcal{E}(\mathbf{a}, \mathit{format})$ is an "encoding" function that maps a group element $\mathbf{a} \in \mathcal{H}$ to a byte string.

  The second argument *format* may be used to choose from one of several possible formats for the encoding of a group element.

  We do not strongly recommend the use of multiple encoding formats, but it is in some cases an already established practice which we need to properly model here.

  We require that the set of all outputs of $\mathcal{E}$ is a subset of some easy-to-recognize, prefix-free language.
- $\mathcal{D}(x)$ is a function that **fails** if $x$ is not a proper encoding of an element of $\mathcal{H}$; otherwise, it returns the group element $\mathbf{a} \in \mathcal{H}$ such that $\mathcal{E}(\mathbf{a}) = x$.

  If a group supports multiple encoding formats, we require that the *format* value used to encode a group element is evident from the encoding itself.
- $\mathcal{E}'(\mathbf{a})$ is a "partial encoding" function that maps a group element $\mathbf{a} \in \mathcal{H}$ to a byte string.

  We require that the set of all outputs of $\mathcal{E}'$ is a subset of some easy-to-recognize, prefix-free language.
- $\mathcal{D}'(x)$ is a function that either **fails** if $x$ is not a proper partial encoding of an element of $\mathcal{H}$; otherwise, it returns the set containing all group elements $\mathbf{a} \in \mathcal{H}$ such that $\mathcal{E}'(\mathbf{a}) = x$. We will assume that the size of this set is bounded by a small constant.

All of the above algorithms should have efficient implementations. The function $\mathcal{D}'$ will never be used by any of the schemes, but the existence of this function is necessary to analyze their security.

We of course assume that arithmetic in $\mathcal{H}$ can be carried out efficiently.

We also assume that we can efficiently test if an element of $\mathcal{H}$ lies in the subgroup $\mathcal{G}$. If all elements in $\mathcal{H}$ of order $\mu$ lie in $\mathcal{G}$, then we can test if $\mathbf{a} \in \mathcal{G}$ by testing if $\mu \cdot \mathbf{a} = \mathbf{0}$. This test is therefore applicable of $\mathcal{H}$ is itself cyclic, or if $\gcd(\mu, \nu) = 1$. For specific groups, there may be more efficient tests of subgroup membership.

This abstraction is meant to be flexible enough to model two important classes of groups: subgroups of $\mathbf{Z}_p^*$, and subgroups of elliptic curves.

## 13.1 Subgroups of $\mathbf{Z}_p^*$

Let $p$ be a prime, and consider the multiplicative group of units modulo $p$, denoted $\mathbf{Z}_p^*$. Let $\mathcal{H}$ denote this group. Let $\mathcal{G}$ denote any prime-order subgroup of $\mathbf{Z}_p^*$. Set $\mu = |\mathcal{G}|$ and $\nu = (p-1)/\mu$. Because $\mathcal{H}$ is itself cyclic, it follows that $\mathcal{G}$ contains all elements of $\mathcal{H}$ whose order divides $\mu$, even if $\gcd(\mu, \nu) \neq 1$. The encoding map $\mathcal{E}$ can be implemented using the function *I2OSP*, where all group elements are encoded as byte strings of length $\lceil \log_{256} p \rceil$. The map $\mathcal{D}$ can be implemented using *OS2IP*. The function $\mathcal{E}'$ is the same as $\mathcal{E}$, and $\mathcal{D}'$ is the same as $\mathcal{D}$.

Note that one can also work with subgroups of arbitrary finite fields, as is done in IEEE P1363.

### 13.2 Subgroups of Elliptic Curves

Let $E$ be an elliptic curve defined over a finite field $\mathbf{F}_q$. Let $\mathcal{H}$ denote this group. Note that $\mathcal{H}$ is not in general cyclic. Let $\mathcal{G}$ denote a prime-order subgroup, and let $\mu$ be its order, and $\nu$ be its index in $\mathcal{H}$. The encoding/decoding maps $\mathcal{E}$ and $\mathcal{D}$ can be implemented using the techniques described in IEEE P1363. Note that these encoding techniques allow for a variety of formats: uncompressed, compressed, and hybrid. The partial encoding map $\mathcal{E}'$ outputs a fixed length byte string encoding of the $x$-coordinate of the point, provided the point is not the "point at infinity"; otherwise, it outputs, say, the all-zero byte string of the same fixed length. The partial decoding map $\mathcal{D}'$ converts the given by string back into an element of $\mathbf{F}_q$, and then solves a polynomial equation to find the set of possible $y$-coordinates (there are at most two).

## 14 Intractability assumptions related to groups

Let

$$Group = (\mathcal{H}, \mathcal{G}, \mathbf{g}, \mu, \nu, \mathcal{E}, \mathcal{D}, \mathcal{E}', \mathcal{D}')$$

as in §13.

### 14.1 The Computational Diffie-Hellman Problem

The Computational Diffie-Hellman (CDH) problem for this group is as follows. On input $(x\mathbf{g}, y\mathbf{g})$, where $x, y \in \{0, \ldots, \mu - 1\}$, compute $xy \cdot \mathbf{g}$. We assume the inputs are random, i.e., that $x$ and $y$ are randomly chosen from the set $\{0, \ldots, \mu - 1\}$.

The CDH assumption is the assumption that this problem is intractable.

Note that in general, it is not feasible to even identify a correct solution to the CDH problem (this is the Decisional Diffie-Hellman problem — see below). In analyzing cryptographic systems, the types of algorithms for solving the CDH that most naturally arise are algorithms that produce a list of candidate solutions to a given instance of the CDH problem. For any algorithm $A$ for the CDH problem that produces a list of length at most $l$, we let $Advantage_{CDH}(A, l)$ denote the probability that this list contains a correct solution to the input problem instance.

Note that in [Sho97], it is shown how to take an algorithm $A$ with $\epsilon = Advantage_{CDH}(A, l)$, and transform this into an algorithm $A'$ that produces a single output that for all inputs is correct with probability $1 - \delta$. The running time of $A'$ is roughly equal to $O(\epsilon^{-1} \log(1/\delta))$ times that of $A$, plus the time to perform

$$O(\epsilon^{-1} l \log(1/\delta) \log \mu + (\log \mu)^2)$$

additional group operations.

It is well known that the CDH problem is "random self reducible."

### 14.2 The Decisional Diffie-Hellman Problem

The Decisional Diffie-Hellman (DDH) problem is as follows.

We define two distributions.

Distribution $\mathbf{R}$ consists of triples $(x\mathbf{g}, y\mathbf{g}, z\mathbf{g})$, where $x, y, z$ are chosen at random from $\{0, \ldots, \mu - 1\}$. Let $X_{\mathbf{R}}$ denote a random variable sampled from this distribution.

Distribution $\mathbf{D}$ consists of triples $(x\mathbf{g}, y\mathbf{g}, z\mathbf{g})$, where $x, y$ are chosen at random from $\{0, \ldots, \mu - 1\}$, and $z = xy \bmod \mu$. Let $X_{\mathbf{D}}$ denote a random variable sampled from this distribution.

The problem is to distinguish these two distributions.

For an algorithm $A$ that outputs either 0 or 1, we define

$$Advantage_{DDH}(A) = |\Pr[A(X_{\mathbf{R}}) = 1] - \Pr[A(X_{\mathbf{D}}) = 1]|.$$

The DDH assumption is that this advantage is negligible for all efficient algorithms.

The DDH problem is "random self-reducible" (see [Sta96] and [NR97]). See [Bon98] and [NR97] for further discussion of the DDH.

### 14.3 The Gap-CDH Problem

The submitters of the *PSEC* scheme have proposed a new computational assumption, called the *gap-CDH assumption.* This is the assumption that it is hard to solve the CDH problem, even in the presence of an oracle for solving the DDH problem.

This assumption is not entirely unreasonable, as it it is easily seen that there is no "black box" reduction from the CDH problem to the DDH problem. This can easily be proven in the "black box group" or "generic group" model of [Sho97].

For any algorithm $A$ that makes at most $q$ queries to a DDH oracle, we define $Advantage_{GapCDH}(A, q)$ to the the probability that $A$ solves a random instance of the CDH problem.

See [OP01] for more details about this assumption.

## 15 *ECIES-KEM*

We present here an encryption scheme that is a slight variant of *ECIES*, and also bears many similarities to *PSEC-3*. What we describe is actually a key encapsulation mechanism, which we call *ECIES-KEM*.

We have to describe the key generation, encryption, and decryption algorithms.

### 15.1 Key Generation

A fully specified group

$$Group = (\mathcal{H}, \mathcal{G}, \mathbf{g}, \mu, \nu, \mathcal{E}, \mathcal{D}, \mathcal{E}', \mathcal{D}')$$

is chosen.

Two additional parameters need to be chosen, which we call *CofactorMode* and *CheckMode*. Each of these parameters take 0/1 values. These modes are used to deal with security problems that can arise when $\nu > 1$. Here are the rules which should be obeyed in setting these modes.

- If $\nu = 1$, then both of these modes should be 0.
- If $\nu > 1$, both modes can be set to 0, provided $\gcd(\mu, \nu) = 1$ and $\nu$ is very small. Note that security in this case degrades by a factor of $\nu$.
- If $\nu > 1$, *CofactorMode* may be set to 1 provided $\gcd(\mu, \nu) = 1$.

- At most one of *CofactorMode* and *CheckMode* should be set to 1.

In addition to *Group*, a key derivation function *KDF* needs to be selected.

Next, a number $x \in \{1, \ldots, \mu - 1\}$ is chosen at random, and the group element $\mathbf{h} = x\mathbf{g}$ is computed.

The public key consists of encodings of *Group* and $\mathbf{h}$, along with an indication of the choice of *KDF*. The precise format of this encoding is not specified here.

The private key consists of the public key, together with the number $x$ and the values *CofactorMode* and *CheckMode*.

## 15.2 Encryption

Recall that for a key encapsulation mechanism, the goal is to produce a ciphertext $C_0$ that is an encryption of a key $K$, where $K$ is a byte string of length $\mathit{KeyLen} = \mathit{ECIES\text{-}KEM.OutputKeyLen}$.

In this scheme, the encryption algorithm may take an optional argument *format* that specifies the format to be used for encoding group elements. The algorithm runs as follows.

1. Choose $r \in \{1, \ldots, \mu - 1\}$ at random.
2. Compute $\tilde{\mathbf{g}} = r\mathbf{g}$ and $\tilde{\mathbf{h}} = r\mathbf{h}$.
3. Output the ciphertext
$$C_0 = \mathcal{E}(\tilde{\mathbf{g}}, \mathit{format}),$$
and the key
$$K = \mathit{KDF}(C_0 \,\|\, \mathcal{E}'(\tilde{\mathbf{h}}), \mathit{KeyLen}).$$

## 15.3 Decryption

The decryption algorithm on input $C_0$ runs as follows.

1. Parse $C_0$ as the encoding of a group element $\tilde{\mathbf{g}} \in \mathcal{H}$. This step fails is $C_0$ is not a proper encoding of an element of $\mathcal{H}$.
2. If $\mathit{CheckMode} = 1$, test if $\tilde{\mathbf{g}} \in \mathcal{G}$; if not, then fail.
3. If $\mathit{CofactorMode} = 1$, set $\hat{\mathbf{g}} = \nu\tilde{\mathbf{g}}$ and $\hat{x} = \nu^{-1}x \bmod \mu$; otherwise, set $\hat{\mathbf{g}} = \tilde{\mathbf{g}}$ and $\hat{x} = x$.
4. Compute $\tilde{\mathbf{h}} = \hat{x}\hat{\mathbf{g}}$.
5. If $\tilde{\mathbf{h}} = \mathbf{0}$, then fail.
6. Output the key
$$K = \mathit{KDF}(C_0 \,\|\, \mathcal{E}'(\tilde{\mathbf{h}}), \mathit{KeyLen}).$$

## 15.4 Some remarks

Using $\mathit{CofactorMode} = 1$ may yield a performance benefit if $\nu$ is fairly small. Note that in this mode, an implementation could simply pre-compute and store the value $\hat{x}$, instead of the value $x$.

## 15.5 Security considerations

This scheme can be proved secure against adaptive chosen ciphertext attack in the random oracle model under the gap-CDH assumption (see §14.3). Here, we model *KDF* as a random oracle.

Indeed, it is straightforward to show that

$$Advantage_{ECIES\text{-}KEM}(A) = O(Advantage_{GapCDH}(A', q_{KDF}))$$

where

- $A'$ is an algorithm with access to a DDH oracle whose running time is about the same as that of $A$,
- $q_{KDF}$ is a bound on the number of random oracle queries, and
- $Advantage_{GapCDH}$ is as defined in §14.3.

This estimate assumes that either $CofactorMode = 1$ or $CheckMode = 1$; otherwise, the security bound is

$$Advantage_{ECIES\text{-}KEM}(A) = O(\nu \cdot Advantage_{GapCDH}(A', q_{KDF})).$$

It can also be proved secure under an appropriate "oracle hashing" assumption, as put forward in the DHAES paper [ABR98].

## 15.6 Compatibility with the IEEE P1363a version of *ECIES*

The key encapsulation mechanism *ECIES-KEM*, when combined with the data encapsulation mechanism *DEM1* described in §10, yields a hybrid encryption scheme that is compatible with the IEEE P1363a version of *ECIES*, provided the choice of group, *KDF*, *MAC*, *SKE* is restricted to be consistent with the IEEE P1363a version.

To remain compatible with IEEE P1363a, we have restricted the group elements $\tilde{\mathbf{g}}$, $\mathbf{h}$, and in particular $\tilde{\mathbf{h}}$ to *not* be the identity. In particular, this means that the partial encoding function $\mathcal{E}'$ is never evaluated at $\mathbf{0}$, which is consistent with IEEE P1363a.

Note that we have made a number of restrictions on the scheme that are not made in the IEEE P1363a version:

1. **Key derivation using $C_0$:** we insist that $C_0$ be passed to *KDF*, whereas this is optional in IEEE P1363a.
2. **No other key derivation parameters:** we do not allow any key derivation parameters, whereas IEEE P1363a allows an arbitrary byte string as a key derivation parameter.
3. **Proper label formatting:** we insist that the input to the *MAC* include the length of the label $L$, whereas this is optional in IEEE P1363a.
4. **No stream cipher option:** we insist on performing data encapsulation using the *DEM1* scheme described in §10, whereas IEEE P1363a allows an alternative mechanism in which *KDF* is used directly as a stream cipher.
5. **No use of "old" cofactor mode:** we insist on using the newer, "compatible" cofactor mode, whereas IEEE P1363a also allows the use of an "old" cofactor mode.

6. **Messages are byte strings:** we insist that messages are byte strings, whereas IEEE P1363a allows these to be bit strings.

7. **Static selection of system parameters:** we insist that all system parameters, including the choice of *KDF*, *SKE*, and *MAC*, be fixed at key generation time, whereas IEEE P1363a allows these to vary dynamically over the lifetime of the public key.

8. $\gcd(\mu, \nu) = 1$ **if both** *CofactorMode* **and** *CheckMode* **are zero:** we insist on this, whereas IEEE P1363a does not.

Each of these restrictions is discussed in turn below. For each restriction, we discuss the rationale for the restriction, and also discuss the the option of easing the restriction so as to achieve greater compatibility with IEEE P1363a. Although in each case, we provide particular arguments for making the recommended restriction, one general argument that applies in all cases is the appeal for *simplicity*: the IEEE P1363a version of *ECIES* provides a fairly bewildering array of options, and it is not clear if all of these options are either desirable or useful.

### 15.6.1 Key derivation using $C_0$

We insist that $C_0$ be passed to *KDF*, whereas this is optional in IEEE P1363a.

We offer two reasons for this requirement.

First, without this requirement, the scheme does not achieve security against adaptive chosen ciphertext attack.

There are a number of simple examples that illustrate why *ECIES* does not achieve this level of security. In particular, it is *malleable*. If the group is an elliptic curve, and the partial encoding function $\mathcal{E}'$ encodes only the $x$-coordinate of a point, then the derived key $K$ is the same if one takes a given ciphertext $C_0$ encoding a point $\tilde{\mathbf{g}}$ and replaces it with an encoding of $-\tilde{\mathbf{g}}$. A similar problem arises if $\nu > 1$ and *CofactorMode* $= 1$ — in this case, one could add to $\tilde{\mathbf{g}}$ a non-zero element whose order divides $\nu$, and one obtains yet again a different ciphertext that decrypts to the same thing. Essentially the same problem arises again if the group supports multiple encoding formats.

Of course, this does not represent a catastrophic failure of the system; it simply illustrates that the definition of adaptive chosen ciphertext security is not met in a strict sense. Indeed, the scheme is still secure in the sense of being only *benignly malleable* (see §2.3), which may be acceptable in many applications. Note, however, that if $\nu > 1$ and both *CofactorMode* and *CheckMode* are zero, then the scheme does not even achieve our weaker notion of benign malleability: it still does not appear to be patently insecure, but it is not clear what useful abstract security properties one can establish for the scheme.

The second reason for our requirement is that including $C_0$ as an input to *KDF* yields a much tighter reduction from the gap-CDH problem. If $q_D$ is the number of decryption requests, and $q_{KDF}$ is the number of random oracle requests, then without the hash of $C_0$, the number of DDH oracle calls that must be made is $q_{KDF} \cdot q_D$, whereas with the hash of $C_0$, this drops to $q_{KDF}$. This security advantage is amplified even further in the multi-user/multi-message setting (see [BBM00]).

The latest draft of IEEE P1363a allows for an optional "DHAES mode," which (among other things) passes $C_0$ to *KDF* in just the same way we have done here. Thus, by making this requirement, the ISO standard would conform to the IEEE P1363a standard. Instead of just allowing it, we strongly recommended that the ISO standard requires it, so as to achieve full non-malleability, and perhaps more importantly, to obtain a much tighter security reduction, especially since the cost of hashing $C_0$ is negligible compared to the cost of the public-key operations. However, it would also be acceptable if the ISO standard allowed a variation of the scheme in which $C_0$ is not hashed, in order to achieve a greater degree of consistency between IEEE P1363a and the ISO standard.

#### 15.6.2 No other key derivation parameters

We do not allow any key derivation parameters, whereas IEEE P1363a allows an arbitrary byte string as a key derivation parameter.

The notion of a key derivation parameter does not fit well with the abstract interface for encryption proposed in this document, and since it is a quite unusual functionality, it seems likely that this is a feature of the IEEE P1363a version of *ECIES* that will quickly atrophy. Indeed, none of the other encryption scheme in IEEE P1363 support an analogous feature. Nevertheless, if this feature is desired, it could be included for backward compatibility's sake.

#### 15.6.3 Proper label formatting

We insist that the input to the *MAC* include the length of the label $L$, whereas this is optional in IEEE P1363a.

In an early draft of the IEEE P1363a version of *ECIES*, the *MAC* is evaluated (in our notation) on the string $c \,\|\, L$, instead of on the string $c \,\|\, L \,\|\, \mathit{I2OSP}(8 \cdot |L|, 8)$, as we have proposed in §10.

The reason we insist on this adding this length information as input to *MAC*, is that without it, the scheme may be malleable.

The problem is that without encoding the length of $L$ in *MAC* input, one can potentially choose any pair of strings $(c', L')$ such that $c' \,\|\, L' = c \,\|\, L$, and then the decryption oracle in a chosen ciphertext attack when supplied with the same ciphertext but with label $L'$ instead of $L$ may leak interesting information about the target message.

This problem was identified and brought to the attention to the IEEE P1363 working group by David Hopwood, as well as through an earlier version of this document. The latest version of the IEEE P1363a draft supports an optional "DHAES mode," which (among other things) provides the same "fix" that we propose here. For compatibility reasons, one might allow variations without the fix, but then require a restriction, such as the restriction that the label must be *empty*, or that all messages are of a length that is fixed for the lifetime of the public key (which would also be consistent with the recommendation below in §15.6.4).

#### 15.6.4 No stream cipher option

We insist on performing data encapsulation using the *DEM1* scheme described in §10, whereas IEEE P1363a allows an alternative mechanism in which *KDF* is used directly as a stream cipher.

The reason we insist on this restriction, is that without it, the scheme is malleable in a very strong sense.

We first describe the "stream cipher option," and then we describe the attack.

Suppose the input to the encryption algorithm is a message $M$ of length $l$, and a label $L$. After the shared Diffie-Hellman key $\tilde{\mathbf{h}}$ is produced, a key derivation function is applied to obtain a string $k \,\|\, k'$, where $k$ has length $l$ and $k'$ has length *MAC.KeyLen*. The ciphertext is

$$C = (C_0, c, \mathit{MAC.eval}(k', c \,\|\, L)),$$

where $c = M \oplus k$ and $C_0$ is the encoding of the ephemeral Diffie-Hellman key $\tilde{\mathbf{g}}$.

Now we describe the attack. Suppose

$$C = (C_0, c, tag)$$

is the encryption of a message $M$ with label $L$, such that

- the length $l$ of $M$ is equal to $l' + MAC.KeyLen$ for $l' > 0$,
- $c = c_1 \| c_2$, where $|c_1| = l'$ and $|c_2| = MAC.KeyLen$
- $M = M_1 \| M_2$, where $|M_1| = l'$ and $|M_2| = MAC.KeyLen$, and
- $M_2$ is known to the attacker.

Then for any byte string $\Delta$ of length $l'$, and any label $\tilde{L}$, the ciphertext

$$\tilde{C} = (C_0, \tilde{c}, MAC.eval(\tilde{k}, \tilde{c} \| \tilde{L})),$$

where

$$\tilde{c} = c_1 \oplus \Delta \text{ and } \tilde{k} = c_2 \oplus M_2,$$

is a valid encryption of $M_1 \oplus \Delta$ with label $\tilde{L}$.

Thus, the scheme is trivially malleable, in a very strong way: we can transform the encryption of $M_1 \| M_2$ with label $L$ into an encryption of $M_1 \oplus \Delta$ with label $\tilde{L}$, for any $\Delta$ and any $\tilde{L}$.

This problem could easily have been avoided if the output of the key derivation function was parsed as $k' \| k$ instead of $k \| k'$. Indeed, if this were done, the security of the data encapsulation method could be proven secure under standard assumptions.

Because of this rather serious flaw in the design of *ECIES*, this mode of data encapsulation was not included in our proposal here, and it is strongly recommended that the ISO version of *ECIES* not allow this mode of data encapsulation.

The only possible circumstances under which the stream cipher option would be acceptable as an option would be if the message length were *fixed* for the lifetime of a public key.

The current IEEE P1363a draft document recommends using the stream cipher option only in applications where the message is short, such as key transport, but the reasons given are efficiency (decryption cannot be performed in a single pass) and the fact that the recommended instantiations of *KDF* have not traditionally been used as stream ciphers, and so there may be unforeseen security problems in their use as such. However, no recommendation is made in IEEE P1363a that the length of the message should be *fixed* for all ciphertexts. Thus, this appears to be an authentic, and perhaps serious, security hole in *ECIES*. The same problem appears in the versions of *ECIES* submitted to ISO, submitted to Crypto-Nessie, and in the draft of ANSI X9.63.[4]

### 15.6.5 No use of "old" cofactor mode

We insist on using the newer, "compatible" cofactor mode, whereas IEEE P1363a also allows the use of an "old" cofactor mode.

The cofactor mode described in this document corresponds to what is called "compatible" cofactor mode in the IEEE P1363a version of *ECIES*. That version of *ECIES* also allows another mode, which we shall call here "old" cofactor mode. "Old" cofactor mode appeared in the *ECIES* submission to ISO, but "compatible" cofactor mode did not. Both of these modes deal quite

[4]This refers to the draft of January 8, 1999.

effectively with the potential problem of small subgroup attacks when $\nu \neq 1$. The advantage of "compatible" cofactor mode over "old" cofactor mode is that the encryption algorithm is oblivious to it in the former, while it needs to be aware of it in the latter. "Old" cofactor mode appears to offer no advantages at all, and so we recommend not including it in the ISO standard. However, we could include it for compatibility reasons, without any great harm.

Note that "compatible" cofactor mode is not "just" an alternative implementation of the decryption algorithm: the behavior of the decryption algorithm is different when using this mode than it is when not, as some ciphertexts that would be rejected without this mode, will not be rejected with this mode.

### 15.6.6 Messages are byte strings

We insist that messages are byte strings, whereas IEEE P1363a allows these to be bit strings.

See §2.1.8 for a discussion of why we work only with byte strings. It is mentioned there that the IEEE P1363a version of *ECIES* allows bit strings, but even that is not entirely true, since the underlying symmetric-key encryption schemes that it currently allows do not support bit strings. The only exception to this is when the "stream cipher option" discussed in §15.6.4 is used, but as we have already argued, this option should anyway not be allowed in the ISO standard.

### 15.6.7 Static selection of system parameters

We insist that all system parameters, including the choice of *KDF*, *SKE*, and *MAC* be fixed at key generation time, whereas IEEE P1363a allows these to vary dynamically over the lifetime of the public key. Although the latter is not recommended in IEEE P1363a, due to possible security problems that may arise from "unintended interactions" of different options, it is nevertheless allowed.

It is the opinion of this author that allowing such flexibility is entirely unacceptable: all hope of a meaningful security analysis vanishes if one allows for this, and there may indeed be real harm that could come from "unintended interactions." This proposal recommends *with the strongest possible urgency* that the ISO standard should *require* all such options to be fixed at key generation time. In addition, all scheme options discussed above, such as not including $C_0$ in the key derivation function, allowing additional key derivation parameters, cofactor mode, check mode, etc., should be determined at key generation time and fixed for the lifetime of public key.

### 15.6.8 $\gcd(\mu, \nu) = 1$ if both *CofactorMode* and *CheckMode* are zero

We insist on this, whereas IEEE P1363a does not.

The reason is that without this restriction, it does not seem possible to reason about the security of the scheme, whereas with this restriction, it is possible.

Without going into all the details, we just note that if $\gcd(\mu, \nu) = 1$, then one can decompose $\mathcal{H}$ as the direct sum of $\mathcal{G}$ and the subgroup $\mathcal{G}' \subset \mathcal{H}$ consisting of all elements of $\mathcal{H}$ whose order divides $\nu$. This decomposition is effective, in the sense that given $\mathbf{a} \in \mathcal{H}$, one can efficiently compute $\mathbf{a}_1 \in \mathcal{G}$ and $\mathbf{a}_2 \in \mathcal{G}'$ such that $\mathbf{a} = \mathbf{a}_1 + \mathbf{a}_2$. The existence of an effective decomposition such as this is critical to the proof of security.

Making this requirement most likely will have little practical impact, since it is very unusual to have $\gcd(\mu, \nu) \neq 1$.

### 15.7 Compatibility with the submitted version of *ECIES*

There are a number of differences between the IEEE P1363a version of *ECIES* and the version of *ECIES* that was submitted to the ISO.

First, and most prominently, the IEEE P1363a version allows subgroups of finite fields as groups in addition to just elliptic curve groups, as in the submitted version.

Second, the IEEE P1363a version allows for a data encapsulation mechanism of the type described in §10 and recommended here, in addition to the "stream cipher option" (see §15.6.4) as in the submitted version. Note that the version of *ECIES* submitted to Crypto-Nessie allows for both modes of operations (although not with the same set of symmetric-key encryption schemes).

Third, the IEEE P1363a version allows for a "compatible" cofactor mode, in addition to the "old" cofactor mode, as in the submitted version (see §15.6.5).

Fourth, the IEEE P1363a version of *ECIES* provides a "DHAES option," which when used, passes $C_0$ to *KDF* (see §15.6.1) and the length of $L$ to *MAC* (see §15.6.3).

In preparing this document, based on discussions with ISO working group members, priority was given to consistency with the IEEE P1363a version, rather than to the submitted version.

## 16 *PSEC-KEM*

We present here a variant of *PSEC-2*. This is a key encapsulation scheme, which we call *PSEC-KEM*, that can be combined with the general hybrid method in §5 to get a full public-key encryption scheme. While the scheme we present here differs in numerous details from the original *PSEC-2*, we believe it is similar in spirit to the *PSEC-2* submission, preserves the main idea of [FO99] on which it is based, and provides very nearly the same security/efficiency trade-off.

### 16.1 Key Generation

A fully specified group

$$Group = (\mathcal{H}, \mathcal{G}, \mathbf{g}, \mu, \nu, \mathcal{E}, \mathcal{D}, \mathcal{E}', \mathcal{D}').$$

Additionally, a key derivation function *KDF* (see §12) should be selected, along with a positive integer *SeedLen*.

Next, a number $x \in \{0, \ldots, \mu - 1\}$ is chosen at random, and the group element $\mathbf{h} = x\mathbf{g}$ is computed.

The public key consists of encodings of *Group* and $\mathbf{h}$, along with an indication of the choice of *KDF* and the value *SeedLen*. The precise format of this encoding is not specified here.

The private key consists of the public key together with $x$.

### 16.2 Encryption

Recall that for a key encapsulation mechanism, the goal is to produce a ciphertext $C_0$ that is an encryption of a key $K$, where $K$ is a byte string of length *KeyLen* = *PSEC-KEM.OutputKeyLen*.

Let *I0* = *I2OSP*$(0, 4)$ and *I1* = *I2OSP*$(1, 4)$.

The encryption algorithm takes an optional argument *format* that specifies the format to be used to encode group elements, and runs as follows.

1. Choose a random byte string $s$ of length *SeedLen*.

2. Compute
$$t = \mathit{KDF}(\mathit{I0} \,\|\, s, \lceil \log_{256} \mu \rceil + 16 + \mathit{KeyLen}),$$
a byte string of length $\lceil \log_{256} \mu \rceil + 16 + \mathit{KeyLen}$.

3. Parse $t$ as $t = u \,\|\, K$, where $|u| = \lceil \log_{256} \mu \rceil + 16$ and $|K| = \mathit{KeyLen}$.

4. Compute $r = \mathit{OS2IP}(u) \bmod \mu$.

5. Compute $\tilde{\mathbf{g}} = r\mathbf{g}$ and $\tilde{\mathbf{h}} = r\mathbf{h}$.

6. Set $\mathit{EG} = \mathcal{E}(\tilde{\mathbf{g}}, \mathit{format})$ and $\mathit{PEH} = \mathcal{E}'(\tilde{\mathbf{h}})$.

7. Compute
$$v = s \oplus \mathit{KDF}(\mathit{I1} \,\|\, \mathit{EG} \,\|\, \mathit{PEH}, \mathit{SeedLen}).$$

8. Output the key $K$ and the ciphertext $C_0 = \mathit{EG} \,\|\, v$.

## 16.3 Decryption

The decryption algorithm takes the secret key as well as a ciphertext $C_0$ as input. It runs as follows.

1. Parse $C_0$ as $C_0 = \mathit{EG} \,\|\, v$, where $\mathit{EG}$ is an encoding a group element $\tilde{\mathbf{g}}$, and $v$ is a byte string of length $\mathit{SeedLen}$. This step may, of course, fail.

2. Compute $\tilde{\mathbf{h}} = x\tilde{\mathbf{g}}$.

3. Set $\mathit{PEH} = \mathcal{E}'(\tilde{\mathbf{h}})$.

4. Compute
$$s = v \oplus \mathit{KDF}(\mathit{I1} \,\|\, \mathit{EG} \,\|\, \mathit{PEH}, \mathit{SeedLen}).$$

5. Compute
$$t = \mathit{KDF}(\mathit{I0} \,\|\, s, \lceil \log_{256} \mu \rceil + 16 + \mathit{KeyLen}),$$
a byte string of length $\lceil \log_{256} \mu \rceil + 16 + \mathit{KeyLen}$.

6. Parse $t$ as $t = u \,\|\, K$, where $|u| = \lceil \log_{256} \mu \rceil + 16$ and $|K| = \mathit{KeyLen}$.

7. Compute $r = \mathit{OS2IP}(u) \bmod \mu$.

8. Compute $\bar{\mathbf{g}} = r\mathbf{g}$.

9. Test if $\bar{\mathbf{g}} = \tilde{\mathbf{g}}$; if not, then fail.

10. Output the key $K$.

## 16.4 Some remarks

Note that in this scheme, we do not have to make an additional check to ensure that $\tilde{\mathbf{g}}$ lies in $\mathcal{G}$ during the decryption process. This is already taken care of by the test in step 9 of the decryption algorithm.

Also note that unlike *ECIES*, a value of $\mathbf{0}$ for $\tilde{\mathbf{h}}$ is perfectly legal, and therefore, the function $\mathcal{E}'$ must be well defined at $\mathbf{0}$. It was felt that making the restriction that $\tilde{\mathbf{h}} \neq \mathbf{0}$ would only complicate the scheme, with no tangible benefit.

### 16.5 Changes from *PSEC-2*

There are a number of substantial differences between *PSEC-KEM* and the *PSEC-2*.

First and foremost is the fact that the above scheme is just a key encapsulation mechanism. As we discussed in §5, using this we can build a hybrid scheme.

The *PSEC-2* submission proposed a different kind of hybrid construction. We would recommend the hybrid construction here above the hybrid construction in the *PSEC-2* for three reasons.

1. One of the goals of this document is to consolidate the various submissions, taking the best ideas from all of them, and obtaining a small set of schemes, each of which offers something unique. To that end, it seems like a good idea to use the same hybrid construction for all schemes.

2. The hybrid construction proposed here has a distinct advantage over the hybrid construction proposed in *PSEC-2*. Namely, it facilitates the implementation of the encryption and decryption algorithms as filters (see §2.1.3). For the original *PSEC-2* construction, this does not seem possible.

3. The hybrid construction proposed here does not rely on random oracles, whereas that in *PSEC-2* does. It is easy enough to build a hybrid scheme without random oracles, assuming the underlying key encapsulation mechanism is secure, so it seems worthwhile to do so. In particular, we want to be able to include schemes, like *ACE-KEM*, that do not use random oracles in their security analysis.

The only disadvantages of our proposed hybrid construction are that the ciphertexts are slightly longer (an additional MAC *tag* is required), and additional code is required for its implementation (the MAC code). We feel that these disadvantages are outweighed by the advantages of conformity with the other schemes, and of facilitating "streaming." This, of course, may be a point of discussion by the working group.

There are some other differences as well. In our scheme, the value $v$ (in our notation) is computed by masking the seed $s$ with a cryptographic hash

$$\mathit{KDF}(\mathit{I1} \,\|\, \mathit{EG} \,\|\, \mathit{PEH}, \mathit{SeedLen}),$$

whereas in *PSEC-2*, $s$ is masked directly with *PEH* — no hash at all. Our scheme thus has potentially more compact ciphertexts than *PSEC-2*. Also, by including *EG* in the hash, we get a a much more efficient security reduction in the multi-user/multi-message model (see [BBM00]), and we also deal properly with the multiple group encoding formats.

A serious criticism of the *PSEC-2* scheme as submitted is that there is no detailed proof of the claimed security theorem, either in the submission or elsewhere in the literature. In fact, there is some doubt as to whether the scheme actually is secure under the stated assumptions. The problem is the way the value $v$ (our notation) is computed in *PSEC-2*. As mentioned above, this is computed as $v = s \oplus \mathit{PEH}$. The only requirement in the scheme is that $\mathit{SeedLen} \leq |\mathit{PEH}|$. However, if $\mathit{SeedLen} < |\mathit{PEH}|$, then the ciphertext contains some of bits of *PEH in the clear*. To prove security of this scheme, then, one would (at least) need to show that one could not compute $\tilde{\mathbf{h}}$ from $\tilde{\mathbf{g}}$ and some of the bits of the partial encoding of $\tilde{\mathbf{h}}$. It would appear that requiring that $\mathit{SeedLen} \geq |\mathit{PEH}|$ solves the problem. Note that the stated requirement that $\mathit{SeedLen} \leq |\mathit{PEH}|$ is apparently not a typographic error, since the examples of *PSEC-2* in the appendix of the submission all have $\mathit{SeedLen} < |\mathit{PEH}|$.

A similar, but more severe, criticism applies to the *PSEC-1* submission. More specifically, in the *PSEC-1* encryption algorithm, the ciphertext contains the XOR of the message with a substring of *PEH*. There is no way the semantic security of this scheme can be based upon the DDH assumption, since the DDH assumption does *not* imply that the bits of an encoding of a group element are pseudo-random.

Also note that our proposed scheme works with any prime-order group, not just subgroups of elliptic curves.

We should also mention that the scheme we have proposed here bears some similarities not only to the *PSEC-2* submission, but also to a very similar scheme presented in [BLK00].

### 16.6 Security considerations

Since this proposed scheme differs significantly from *PSEC-2* and other schemes in the literature, we sketch a security proof in the random oracle model assuming the CDH (see §14.1).

We view *KDF* as a random oracle. Note that all relevant inputs to *KDF* start with either a "zero word" or a "one word." This effectively gives us two independent random oracles,

$$\begin{aligned} H_0 &: \mathbf{B}^{SeedLen} \to \mathbf{B}^{\lceil \log_{256} \mu \rceil + 16 + KeyLen}, \\ H_1 &: \mathcal{E}(\mathcal{H}) \times \mathcal{E}'(\mathcal{H}) \to \mathbf{B}^{SeedLen}. \end{aligned}$$

Here, $\mathbf{B}$ denotes the set of *bytes*. Also, $\mathcal{E}(\mathcal{H})$ denotes the set of all encodings of elements in $\mathcal{H}$, using all formats, and $\mathcal{E}'(\mathcal{H})$ denotes the set of all partial encodings of elements in $\mathcal{H}$. In the security analysis, we shall replace the calls to *KDF* by appropriate queries to $H_0$ and $H_1$.

Consider an adversary $A$ that makes $q_D$ calls to the decryption oracle, $q_0$ calls to $H_0$ and $q_1$ calls to $H_1$.

Let $\mathbf{G}_0$ be the original attack game, and let $S_0$ be the event that the adversary correctly guesses the hidden bit $b$ in this game (see §3). We shall define a sequence of attack games $\mathbf{G}_1, \mathbf{G}_2, \ldots, \mathbf{G}_k$. Each of these games should be viewed as operating on the same underlying probability space — only the rules for how certain random variables are computed differ. In each game $\mathbf{G}_i$, $1 \le i \le k$, there will be an event $S_i$ corresponding to $S_0$. We shall show that for all $1 \le i \le k$, that the difference $|\Pr[S_i] - \Pr[S_{i-1}]|$ is negligible, and moreover, it will be evident that in the last game, $\Pr[S_k] = 1/2$. This will imply that $Advantage_{PSEC\text{-}KEM}(A)$, which is equal to $|\Pr[S_0] - 1/2|$, is negligible.

We adopt the following convention. For an arbitrary ciphertext $C_0$, we denote by

$$EG, v, \tilde{\mathbf{g}}, \tilde{\mathbf{h}}, PEH, s, t, u, K, r, \bar{\mathbf{g}},$$

the values computed by the decryption algorithm on this ciphertext. Some of these may be undefined if the algorithm would fail before the value was computed. We also denote the target ciphertext $C_0^*$, and define corresponding values $EG^*, v^*, \tilde{\mathbf{g}}^*, \ldots$.

We classify ciphertexts $C_0$ submitted to the encryption oracle as follows:

**Type I** $\tilde{\mathbf{g}} \ne \tilde{\mathbf{g}}^*$;

**Type II** $EG = EG^*$;

**Type III** $\tilde{\mathbf{g}} = \tilde{\mathbf{g}}^*$, but $EG \ne EG^*$.

Note that all ciphertexts $C_0$ submitted to the decryption oracle *before* the encryption oracle has been invoked are classified as Type I. Notice that Type III ciphertexts can arise only if the group supports multiple encoding formats.

Let $\mathcal{S}$ denote the set of points $s$ at which the oracle $H_0$ has been queried either *(i)* directly by the adversary, or *(ii)* by a Type III decryption oracle invocation. The set $\mathcal{S}$ grows over time, as more queries to $H_0$ are made. For any byte string $s$ of length *SeedLen*, we define $\rho(s)$ to be the number obtained by taking the first $\lceil \log_{256} \mu \rceil + 16$ of $H_0(s)$, converting to an integer, and reducing mod $\mu$.

The following trivial lemma will streamline our arguments.

**Lemma 1** *Let $E$, $E'$, and $F$ be events defined on a probability space such that* $\Pr[E \wedge \neg F] = \Pr[E' \wedge \neg F]$. *Then we have*

$$|\Pr[E] - \Pr[E']| \leq \Pr[F].$$

The proof is a simple calculation, which we omit.

We now define our sequence of games $\mathbf{G}_1, \mathbf{G}_2, \ldots$.

**Game $\mathbf{G}_1$.** We modify the decryption oracle as follows. If the adversary submits a Type II ciphertext $C_0$, then in game $\mathbf{G}_1$, we summarily reject $C_0$, without executing the decryption algorithm at all.

Let $F_1$ be the event that in game $\mathbf{G}_1$ such a ciphertext is rejected that would not have been rejected under the rules of game $\mathbf{G}_0$. Since these two games proceed identically until $F_1$ occurs, we have $\Pr[S_0 \wedge \neg F_1] = \Pr[S_1 \wedge \neg F_1]$, and applying Lemma 1 with $(S_0, S_1, F_1)$, we have $|\Pr[S_0] - \Pr[S_1]| \leq \Pr[F_1]$.

So it suffices to bound $\Pr[F_1]$. Consider a Type II ciphertext $C_0$ submitted to the decryption oracle in game $\mathbf{G}_1$. Since $C_0 \neq C_0^*$, we must have $v \neq v^*$, which implies $s \neq s^*$. To accept under the rules of game $\mathbf{G}_0$, we must have $r = r^*$.

To make this happen, the adversary must find an input $s \neq s^*$ to $H_0$ such that $\rho(s) = r^*$. Thus, $\Pr[F_1] \leq (q_0 + q_D)\mu^{-1}(1 + 2^{-128})$. The factor $(1 + 2^{-128})$ comes from the fact that the value $r$ is not exactly uniformly distributed over $\{0, \ldots, \mu - 1\}$.

So we have

$$|\Pr[S_0] - \Pr[S_1]| \leq (q_0 + q_D)\mu^{-1}(1 + 2^{-128}). \tag{1}$$

**Game $\mathbf{G}_2$.** In this game, we modify the decryption oracle as follows. Suppose a Type I ciphertext $C_0$ is submitted, and suppose that $s \notin \mathcal{S}$. Then we summarily reject this ciphertext, without ever proceeding past step 4 of the decryption algorithm.

Note that in this game, Type I and II decryption oracle invocations never evaluate $H_0$ at points not already in $\mathcal{S}$.

Let $F_2$ be the event that in game $\mathbf{G}_2$ such a ciphertext is rejected that would not have been rejected under the rules of game $\mathbf{G}_1$. These two games proceed identically until $F_2$ occurs, and so $\Pr[S_1 \wedge \neg F_2] = \Pr[S_2 \wedge \neg F_2]$, and applying Lemma 1 to $(S_1, S_2, F_2)$, we have $|\Pr[S_1] - \Pr[S_2]| \leq \Pr[F_2]$.

So it suffices to bound $\Pr[F_2]$. Consider a ciphertext $C_0$ as above is submitted to the decryption oracle in game $\mathbf{G}_2$. On the one hand, if the encryption oracle was previously invoked and $s = s^*$, then under the rules of game $\mathbf{G}_1$, we would certainly reject $C_0$, since $\tilde{\mathbf{g}} \neq \tilde{\mathbf{g}}^*$. On the other hand, if the decryption oracle was not previously invoked or it was but $s \neq s^*$, then $H_0$ was never queried at $s$ either by the encryption oracle, the decryption oracle, or the adversary, and so the value $r$ is independent of everything in the adversary's view. It follows that the probability that this ciphertext would not be rejected under the rules of game $\mathbf{G}_1$ is at most $\mu^{-1}(1 + 2^{-128})$.

From this, it follows that $\Pr[F_1] \leq q_D \mu^{-1}(1 + 2^{-128})$, and therefore,

$$|\Pr[S_1] - \Pr[S_2]| \leq q_D \mu^{-1}(1 + 2^{-128}). \tag{2}$$

**Game $\mathbf{G}_3$.** We make another modification to the decryption oracle. In this new game, we process all Type I ciphertexts $C_0$ as follows. If $\tilde{\mathbf{g}}$ is not equal to $\rho(s')\mathbf{g}$ for any $s' \in \mathcal{S}$, then we reject without any further processing. Otherwise, if $\tilde{\mathbf{g}} = \rho(s')\mathbf{g}$ for some $s' \in \mathcal{S}$, we compute $\tilde{\mathbf{h}} = \rho(s')\mathbf{h}$, and proceed to decrypt just as in game $\mathbf{G}_2$, but starting with step 3 of the decryption algorithm.

We argue that games $\mathbf{G}_2$ and $\mathbf{G}_3$ are identical.

Consider first a ciphertext for which $\tilde{\mathbf{g}}$ is not equal to $\rho(s')\mathbf{g}$ for any $s' \in \mathcal{S}$. This ciphertext would have anyway been rejected under the rules in game $\mathbf{G}_2$. To see this, let $\tilde{\mathbf{g}} = \hat{r}\mathbf{g}$, where $\hat{r} \in \{0, \ldots, \mu - 1\}$. Now, $\hat{r} \neq \rho(s')$ for any $s' \in \mathcal{S}$. Consider the value $s$. If $s \in \mathcal{S}$, then we would reject under the rules in game $\mathbf{G}_2$, since the test in step 9 would fail; otherwise, if $s \notin \mathcal{S}$, we would also reject under the rules in game $\mathbf{G}_2$, since the special rejection rule introduced in game $\mathbf{G}_2$ would apply.

Next, consider the case where $\tilde{\mathbf{g}} = \rho(s')\mathbf{g}$ for some $s' \in \mathcal{S}$. It is clear that in this case, decryption proceeds exactly as in game $\mathbf{G}_2$.

So we have

$$\Pr[S_3] = \Pr[S_2]. \tag{3}$$

**Game $\mathbf{G}_4$.** We modify game $\mathbf{G}_3$ to obtain an equivalent game $\mathbf{G}_4$. This rather technical step is a "bridging" step that will facilitate the analysis of more drastic modifications in game $\mathbf{G}_5$.

In game $\mathbf{G}_4$, we introduce

- a random byte string $s^+$ of length *SeedLen*,
- a random byte string $u^+$ of length $\lceil \log_{256} \mu \rceil + 16$,
- a random byte string $K^+$ of length *KeyLen*, and
- a random *oracle*

$$h^+ : \mathcal{E}(\mathcal{H}) \to \mathbf{B}^{\mathit{SeedLen}}.$$

Game $\mathbf{G}_4$ is identical to game $\mathbf{G}_3$, except that we apply the following special rules:

**R1:** In the encryption oracle, we perform the following steps:

1. Set $r^+ = \mathit{OS2IP}(u^+) \bmod \mu$.
2. Compute $\tilde{\mathbf{g}}^* = r^+\mathbf{g}$.
3. Set $\mathit{EG}^* = \mathcal{E}(\tilde{\mathbf{g}}^*, \mathit{format})$.
4. Compute $v^* = s^+ \oplus h^+(\mathit{EG}^*)$.
5. Output the key $K^+$ and the ciphertext $C_0^* = \mathit{EG}^* \,\|\, v^*$.

**R2:** In the decryption oracle, when processing a Type III ciphertext, we use the value $h^+(\mathit{EG})$ in step 4, instead of $H_1(\mathit{EG}, \mathit{PEH})$.

**R3:** Whenever the oracle $H_0$ is queried — by either the adversary or a Type III decryption oracle — at $s^+$ we respond with $u^+ \,\|\, K^+$, instead of $H(s^+)$.

**R4:** Whenever the oracle $H_1$ is queried — by either the adversary or a Type I decryption oracle — at a point $(\mathit{EG}, \mathit{PEH})$, where $\mathit{EG}$ is an encoding of $\tilde{\mathbf{g}}^*$ and $\mathit{PEH}$ is a partial encoding of $x\tilde{\mathbf{g}}^*$, we respond with $h^+(\mathit{EG})$ instead of $H_1(\mathit{EG}, \mathit{PEH})$.

It is clear that games $\mathbf{G}_3$ and $\mathbf{G}_4$ are completely equivalent, since we have consistently replaced one set of random variables by another set of identically distributed random variables. In particular,

$$\Pr[S_3] = \Pr[S_4]. \tag{4}$$

**Game $\mathbf{G}_5$.** Game $\mathbf{G}_5$ is the same as game $\mathbf{G}_4$, except that we drop rules **R3** and **R4**, while retaining **R1** and **R2**.

Note that in this game, we do not use the secret key of the cryptosystem at all. Also note that the ciphertext $C_0^*$ is no longer a valid ciphertext in general, nor does it hold in general that $s^* = s^+$, or that $t^* = u^+ \parallel K^+$, since the random oracles are no longer consistent with the modifications made in the encryption oracle. Indeed, $K^+$ and hence the hidden bit $b$ are independent of the adversary's view in game $\mathbf{G}_5$. The string $s^+$ is also independent of the adversary's view. Further, the behavior of Type III decryption oracle queries are also not consistent with the random oracles.

Despite these differences, however, games $\mathbf{G}_4$ and $\mathbf{G}_5$ proceed identically until the string $s^+$ appears in $\mathcal{S}$ or either the adversary or a Type I decryption oracle invocation queries $H_1$ on inputs $(EG, PEH)$, where $EG$ is an encoding of $\tilde{\mathbf{g}}^*$ and $PEH$ is the partial encoding of $x\tilde{\mathbf{g}}^*$.

Let $F_{5a}$ be the event that in game $\mathbf{G}_5$, the string $s^+$ appears in $\mathcal{S}$ at some point in time. Let $F_{5b}$ be the event that either the adversary or a Type I decryption oracle invocation queries $H_1$ on inputs $(EG, PEH)$, where $EG$ is an encoding of $\tilde{\mathbf{g}}^*$ and $PEH$ is the partial encoding of $x\tilde{\mathbf{g}}^*$. Let $F_5 = F_{5a} \vee F_{5b}$.

Since games $\mathbf{G}_4$ and $\mathbf{G}_5$ proceed identically until the point where $F_5$ occurs, we have $\Pr[S_4 \wedge \neg F_5] = \Pr[S_5 \wedge \neg F_5]$. Applying Lemma 1 with $(S_4, S_5, F_5)$, we have $|\Pr[S_4] - \Pr[S_5]| \leq \Pr[F_5]$.

Since $s^+$ is independent of the adversary's view, we have

$$\Pr[F_{5a}] \leq (q_0 + q_D)2^{-\mathit{SeedLen}}.$$

Now, $\Pr[F_{5b}]$ is bounded by $(1 + 2^{-128})$ times the probability that an adversary $A'$ — running in expected time nearly the same as the running time of the original adversary $A$ — can construct a list of $O(q_1 + q_D)$ group elements, one of which contains a solution to a given instance of the CDH problem.

This algorithm runs by taking a random instance $(\mathbf{g}, \mathbf{h}, \tilde{\mathbf{g}}^+)$ of the CDH problem as input, and runs $A$ against a slightly modified version of game $\mathbf{G}_5$. In this modified game, we use the given values $\mathbf{g}, \mathbf{h}$ to form the public key in game $\mathbf{G}_5$, Also, we use the given value $\tilde{\mathbf{g}}^+$, instead of deriving it from $u^+$ (note that $u^+$ is not used anywhere else in game $\mathbf{G}_5$). Finally, to implement this algorithm, we simulate the random oracles in the usual way, using standard hash table techniques. We also use standard hash table techniques to implement the Type I decryption oracle queries, as modified in game $\mathbf{G}_3$. The factor $(1 + 2^{-128})$ comes from the fact that the distribution of $\tilde{\mathbf{g}}+$ in game $\mathbf{G}_5$ is slightly non-uniform, whereas we assume the corresponding value in the CDH instance is uniformly distributed.

From this, it follows that

$$\begin{aligned} |\Pr[S_4] - \Pr[S_5]| \;\leq\; & \mathit{Advantage}_{\mathit{CDH}}(A', O(q_1 + q_D))(1 + 2^{-128}) + \\ & (q_0 + q_D)2^{-\mathit{SeedLen}}, \end{aligned} \tag{5}$$

where $\mathit{Advantage}_{\mathit{CDH}}$ is as defined in §14.1.

It is also clear that in game $\mathbf{G}_5$, the hidden bit $b$ is independent of all values directly or indirectly accessible to the adversary. Hence,

$$\Pr[S_5] = 1/2. \tag{6}$$

Putting together (1), (2), (3), (4), (5), (6), we obtain

$$\begin{aligned} \mathit{Advantage}_{\mathit{PSEC\text{-}KEM}}(A) \;\le\; & (q_0 + 2q_D)\mu^{-1}(1 + 2^{-128}) + \\ & \mathit{Advantage}_{\mathit{CDH}}(A', O(q_1 + q_D))(1 + 2^{-128}) + \\ & (q_0 + q_D)2^{-\mathit{SeedLen}}. \end{aligned} \tag{7}$$

## 17 *ACE-KEM*

In this section, we present a variant of the *ACE-Encrypt* submission. Several changes were made to the original submission, so that the resulting scheme fits into our frameworks for hybrid and Diffie-Hellman-based encryption. This variant is a key encapsulation mechanism that we call *ACE-KEM*.

### 17.1 Key Generation

A fully specified group

$$\mathit{Group} = (\mathcal{H}, \mathcal{G}, \mathbf{g}, \mu, \nu, \mathcal{E}, \mathcal{D}, \mathcal{E}', \mathcal{D}')$$

is chosen. In what follows, we let $\mathbf{g}_1 = \mathbf{g}$.

An additional parameter, *CofactorMode*, must be specified. This parameter takes the value 0 or 1. Here are the rules which should be obeyed in setting this parameter.

- If $\nu = 1$, then *CofactorMode* should be 0.
- If $\nu > 1$, *CofactorMode* may be set to 1 provided $\gcd(\mu, \nu) = 1$.

In addition to *Group*, a hash function *Hash* (see §11) and key derivation function *KDF* (see §12) must be chosen. It is required that and that *Hash.OutputLen* $< \log_{256} \mu$.

Since we want *KDF* to be a good entropy smoothing function, one should select either *KDF3* or *KDF4*. As discussed in §12, the functions *KDF1* and *KDF2* are not recommended.

Next, numbers $w, x, y, z \in \{0, \ldots, \mu - 1\}$ are chosen at random, and the group elements

$$\mathbf{g}_2 = w \cdot \mathbf{g}_1,\ \mathbf{c} = x \cdot \mathbf{g}_1,\ \mathbf{d} = y \cdot \mathbf{g}_1,\ \mathbf{h} = z \cdot \mathbf{g}_1$$

are computed.

The public key consists of encodings of *Group*, the group elements $\mathbf{g}_2, \mathbf{c}, \mathbf{d}, \mathbf{h}$, and an indication of the choice of *Hash* and *KDF*. The precise format of this encoding is not specified here.

The private key consists of the public key, together with the numbers $w, x, y, z$.

### 17.2 Encryption

Recall that for a key encapsulation mechanism, the goal is to produce a ciphertext $C_0$ that is an encryption of a key $K$, where $K$ is a byte string of length *KeyLen* = *ACE-KEM.OutputKeyLen*.

In addition to the recipient's public key, the encryption algorithm takes an optional *format* argument, which is used to specify the format for group element encodings.

The encryption scheme works as follows.

1. Choose $r \in \{0, \ldots, \mu - 1\}$.
2. Compute group elements

$$\mathbf{u}_1 = r \cdot \mathbf{g}_1,\ \mathbf{u}_2 = r \cdot \mathbf{g}_2,\ \tilde{\mathbf{h}} = r \cdot \mathbf{h}.$$

3. Compute the byte strings

$$EU1 = \mathcal{E}(\mathbf{u}_1, format), \ EU2 = \mathcal{E}(\mathbf{u}_2, format).$$

4. Compute the number

$$\alpha = OS2IP(Hash.eval(EU1 \,\|\, EU2)).$$

5. Compute the number

$$r' = \alpha \cdot r \bmod \mu.$$

6. Compute the group element

$$\mathbf{v} = r \cdot \mathbf{c} + r' \cdot \mathbf{d}.$$

7. Output the ciphertext

$$C_0 = EU1 \,\|\, EU2 \,\|\, \mathcal{E}(\mathbf{v}, format)$$

and the key

$$K = KDF(EU1 \,\|\, \mathcal{E}'(\tilde{\mathbf{h}}), KeyLen).$$

### 17.3 Decryption

The decryption algorithm takes as input a ciphertext $C_0$ along with the private key.

1. Parse the ciphertext as $EU1 \,\|\, EU2 \,\|\, EV$, where $EU1$ encodes the group element $\mathbf{u}_1$, $EU2$ encodes the group element $\mathbf{u}_2$, and $EV$ encodes the group element $\mathbf{v}$. If this step fails, then fail.

   Also, one must check that *EU1*, *EU2*, and *EV* are all encoded using the same *format*; if not, then fail.

2. If $CofactorMode = 1$, set

$$\hat{\mathbf{u}}_1 = \nu \cdot \mathbf{u}_1, \ \hat{w} = \nu^{-1} w \bmod \nu, \ \hat{x} = \nu^{-1} x \bmod \nu, \ \hat{y} = \nu^{-1} y \bmod \nu, \ \hat{z} = \nu^{-1} z \bmod \nu;$$

   otherwise, set

$$\hat{\mathbf{u}}_1 = \mathbf{u}_1, \ \hat{w} = w, \ \hat{x} = x, \ \hat{y} = y, \ \hat{z} = z.$$

3. If $CofactorMode \neq 1$ and $\nu > 1$, test if $\mathbf{u}_1 \in \mathcal{G}$. If not, then fail.

4. Compute the number

$$\alpha = OS2IP(Hash.eval(EU1 \,\|\, EU2))$$

5. Compute the number

$$t = \hat{x} + \hat{y}\alpha \bmod \mu.$$

6. Test if

$$w \cdot \hat{\mathbf{u}}_1 = \mathbf{u}_2 \text{ and } t \cdot \hat{\mathbf{u}}_1 = \mathbf{v}.$$

   If not, then fail.

7. Compute the group element

$$\tilde{\mathbf{h}} = \hat{z} \cdot \hat{\mathbf{u}}_1.$$

8. Output the key

$$K = KDF(EU1 \,\|\, \mathcal{E}'(\tilde{\mathbf{h}}), KeyLen).$$

### 17.4 Some remarks

For security reasons, one should always perform all of the computations in step 6 of the decryption algorithm; otherwise, some "timing" information could be gained by the adversary that is not available to it in the formal proof of security. Note, however, that we know of no actual attack based on such timing information, nor is such an attack at all likely.

Also note that unlike *ECIES*, a value of $\mathbf{0}$ for $\tilde{\mathbf{h}}$ is perfectly legal, and therefore, the function $\mathcal{E}'$ must be well defined at $\mathbf{0}$. It was felt that making the restriction that $\tilde{\mathbf{h}} \neq \mathbf{0}$ would only complicate the scheme, with no tangible benefit.

Using $\mathit{CofactorMode} = 1$ may yield a performance benefit if $\nu$ is fairly small. Note that in this mode, an implementation could simply pre-compute and store the values $\hat{w}, \hat{x}, \hat{y}, \hat{z}$, instead of the values $w, x, y, z$.

### 17.5 Security considerations

This scheme differs in only very minor ways from schemes that have been rigorously analyzed in the literature. It most closely resembles the variation of the Cramer-Shoup scheme discussed in detail in [Sho00].

The security of the scheme is based on the DDH (see §14.2), and a few other *specific* assumptions about the hash and key derivation functions. The security reduction is quite tight. One can easily verify the following, using following the line of reasoning in [CS98] and [Sho00].

$$
\begin{aligned}
\mathit{Advantage}_{\mathit{ACE\text{-}KEM}}(A) \quad = O( \quad & \mathit{Advantage}_{\mathit{DDH}}(A_1) + \\
& \mathit{Advantage}_{\mathit{Hash}}(A_2) + \\
& \mathit{Advantage}_{\mathit{KDF}}(A_3) + \\
& q_D \cdot \mu^{-1} \ ),
\end{aligned}
$$

where:

- $A_1, A_2, A_3$ denote adversaries that run in time essentially the same as $A$.
- $\mathit{Advantage}_{\mathit{DDH}}$ is as defined in §14.2.
- $\mathit{Advantage}_{\mathit{Hash}}(A)$ denotes the probability that an adversary $A$, given encodings $\mathit{EU1}^*$ and $\mathit{EU2}^*$ of two independent, random elements in $\mathcal{G}$, can find encodings $\mathit{EU1}$ and $\mathit{EU2}$ of elements in $\mathcal{G}$, such that $(\mathit{EU1}, \mathit{EU2}) \neq (\mathit{EU1}^*, \mathit{EU2}^*)$, but

  $$\mathit{Hash.eval}(\mathit{EU1} \,\|\, \mathit{EU2}) = \mathit{Hash.eval}(\mathit{EU1}^* \,\|\, \mathit{EU2}^*).$$

  If the group supports multiple encodings, the adversary can choose the format it wants when $\mathit{EU1}^*$ and $\mathit{EU2}^*$ are generated; furthermore, the adversary may choose to use the same or different formats in its choice of $\mathit{EU1}$ and $\mathit{EU2}$; however, $\mathit{EU1}^*$ and $\mathit{EU2}^*$ must be encoded using the same format, and the same holds for $\mathit{EU1}$ and $\mathit{EU2}$.

  If $\mathit{CofactorMode} = 1$, then the adversary may choose $\mathit{EU1}$ to be an encoding of an element of $\mathcal{H}$ that does not necessarily lie in $\mathcal{G}$.

  Note that this problem is a second-preimage collision problem, which is generally believed to be a much harder problem to solve than the problem of finding an arbitrary pair of colliding inputs.

- $Advantage_{KDF}(A)$ denotes the advantage that an adversary $A$ has in distinguishing between the following two distributions. Let $\mathbf{u}_1$ and $\tilde{\mathbf{h}}$ be independent, random elements of $\mathcal{G}$, and let *EU1* be an encoding of $\mathbf{u}_1$. Let $R$ be a random byte string of length *KeyLen*. The first distribution is $(R, \mathit{EU1})$, and the second is $(\mathit{KDF}(\mathit{EU1} \,\|\, \mathcal{E}'(\tilde{\mathbf{h}}), \mathit{KeyLen}), \mathit{EU1})$.

- $q_D$ bounds the number of decryption oracle queries made by the adversary $A$.

The "O" above represents a very small constant, which we have not computed exactly.

### 17.6 Further remarks

#### 17.6.1 Random oracles and interactive assumptions

We emphasize that this scheme can be proved secure under reasonable intractability assumptions, without resorting to either the random oracle heuristic, and without using "interactive" intractability assumptions as in done in [ABR98].

We stress that a proof of security in the random oracle model *is not* a proof with "just another assumption." One is not assuming a hash function is a random function, since this assumption is patently false. The random oracle model is a *heuristic*, and a proof of security in the random oracle model does not directly imply anything about the security of a system "in the real world."

We also stress that interactive intractability assumptions, like in [ABR98], are qualitatively much stronger than standard intractability assumptions. Indeed, it can be argued that the main activity of theoretical cryptography is to show that breaking a cryptosystem via some kind of subtle, interactive attack is at least as hard as solving some specific, non-interactive problem.

*ACE-KEM* can also be proved secure in the random oracle model under the CDH assumption (see [Sho00]), although the reduction is not nearly as tight as for *PSEC-KEM*. Indeed, the tightness of the reduction for *PSEC-KEM* and the efficiency of *PSEC-KEM* are the main reasons for including *PSEC-KEM* in this proposal.

#### 17.6.2 *ACE-KEM* and *ECIES-KEM*

One should also note that *ACE-KEM* is no less secure than *ECIES-KEM* in a very strong sense. Indeed, assuming the two cryptosystems use the same parameters, then one can show that any adversary $A$ that breaks *ACE-KEM* can be converted into an adversary $A'$ with about the same running time that breaks *ECIES-KEM* with the same advantage.

To see this, consider an *ECIES-KEM* public key containing the group element $\mathbf{h}$. Upon obtaining this public key, $A'$ generates $w, x, y$ at random modulo $\mu$, and then chooses $w, x, y \in \{0, \dots, \mu - 1\}$ at random, and constructs the *ACE-KEM* public key $(\mathbf{g}_2, \mathbf{c}, \mathbf{d}, \mathbf{h})$, where $\mathbf{g}_2 = w\mathbf{g}$, $\mathbf{c} = x\mathbf{g}$, and $\mathbf{d} = y\mathbf{g}$. $A'$ then runs adversary $A$ using this public key.

Now, whenever the adversary $A$ makes a decryption oracle query, then knowing $w, x, y$, adversary $A'$ performs the extra validity tests of *ACE-KEM*, and if these pass, it uses the decryption oracle of *ECIES-KEM* to obtain the decrypted symmetric key, giving this to $A$.

When $A$ invokes the encryption oracle for *ACE-KEM*, $A'$ invokes the encryption oracle for *ECIES-KEM*, obtaining an encoding of a group element $\mathbf{u}_1^*$. Then using $w, x, y$, $A'$ easily constructs the remaining components of a corresponding *ACE-KEM* ciphertext, and gives this to $A$.

One needs to check that $A'$ carries out a legal chosen ciphertext attack, i.e., that $A'$ never attempts to submit the target ciphertext to the decryption oracle subsequent to the invocation of the encryption oracle. But this follows easily from the following claim: for any two *valid* *ACE-KEM* ciphertexts $C_0 = \mathit{EU1} \,\|\, \mathit{EU2} \,\|\, \mathit{EV}$ and $C_0^+ = \mathit{EU1}^+ \,\|\, \mathit{EU2}^+ \,\|\, \mathit{EV}^+$, if $\mathit{EU1} = \mathit{EU1}^+$, then $C_0 = C_0^+$.

Note that this claim relies on the fact that the validity test for a ciphertext $C_0$ as above ensures that *EU1*, *EU2*, and *EV* are encoded using the same format. If this were not done, then simply by replacing *EU2* or *EV* by a different encoding of the same group element, one would violate the above claim.

When $A$ terminates and outputs a bit $\hat{b}$, $A'$ also terminates and outputs the same thing.

It is easily seen that this simulation is perfect, and that whatever advantage $A$ has in breaking *ACE-KEM*, $A'$ has the same advantage in breaking *ECIES-KEM*.

We have left one detail out of the above proof: in *ECIES-KEM*, the Diffie-Hellman public keys may not be zero, whereas in *ACE-KEM*, they may be. We leave it to the reader to adjust the above proof to accommodate this detail.

### 17.6.3 *ACE-Encrypt* and *ACE-KEM*

We outline the major differences between *ACE-Encrypt* and *ACE-KEM*.

- We have generalized the algorithm to work with an arbitrary, abstract group, and to work with an arbitrary message authentication code and symmetric key encryption scheme.

- We have chosen not to use the rather specialized universal one-way hash function to compute the quantity $\alpha$. Instead, we use a standard cryptographic hash, and make a specific — but reasonable — "second preimage collision resistance" assumption.

  The proposed standard need not necessarily preclude the possibility of using such a specialized hash function, so long as we allow such a hash to have a variable length key that is stored in the public key.

- We have chosen not to use the rather specialized entropy-smoothing hash function to derive the key $K$. Instead, we again use a standard cryptographic hash, and make a specific — but again, reasonable — "entropy smoothing" assumption.

  The proposed standard need not necessarily preclude the possibility of using such a specialized hash function, so long as we allow such a hash to have a variable length key that is stored in the public key.

# 18 *RSA-OAEP*

## 18.1 Message encoding functions

*OAEP-EME* is a fully specified version of Bellare and Rogaway's original OAEP scheme for message encoding [BR94].

In general, a message encoding scheme *EME* of this type specifies two algorithms:

- *EME.Encode*$(M, L, ELen)$ takes as input a message $M$ and a label $L$, and an output length *ELen*. Here, $M$ and $L$ are byte strings whose lengths are bounded, as described below. It outputs a byte string $E$ of length *ELen*.

- *EME.Decode*$(E, L)$ takes as input a byte string $E$ and a label $L$. It attempts to find a message $M$ such that *EME.Encode*$(M, L, |E|) = E$. It returns $M$ if such an $M$ exists, and otherwise fails.

In addition to this, the mechanism should specify a bound *EME.Bound* such that when *EME.Encode*$(M, L, ELen)$ is invoked, the condition $|M| \leq ELen - EME.Bound$ should hold; if not, the encoding algorithm fails. Additionally, the encoding algorithm may also fail if $|L|$ exceeds some (very large) implementation-defined bound.

The algorithm *EME.Encode* will in general be probabilistic, so that the same message can be encoded in a number of ways.

## 18.2 OAEP-EME

We now describe *OAEP-EME*.

The scheme is parameterized by a hash function *Hash* (see §11) and a key derivation function *KDF* (see §12). Current standards, as well as the *RSA-OAEP* submission to ISO, recommend the use of the function *KDF1* using *Hash*. Let $HLen = Hash.OutputLen$.

The quantity *OAEP-EME.Bound* is defined as

$$OAEP\text{-}EME.Bound = 2 \cdot HLen + 1.$$

### 18.2.1 Encoding function

The algorithm *OAEP-EME.Encode*$(M, L, ELen)$ runs as follows:

1. Check that $|M| <= ELen - 2 \cdot HLen - 1$; if not, then fail.
2. Generate a random byte string $r$ of length *HLen*.
3. Let *pad* be the byte string of length $ELen - |M| - 2 \cdot HLen$ consisting of a sequence of 0-bytes, followed by a single 1-byte.
4. Set $x = Hash.eval(L) \,\|\, pad \,\|\, M$.
5. Set $s = KDF(r, ELen - HLen) \oplus x$.
6. Set $t = KDF(s, HLen) \oplus r$.
7. Output $E = t \,\|\, s$.

### 18.2.2 Decoding function

The algorithm *OAEP-EME.Decode*$(E, L)$ runs as follows.

1. Let $ELen = |E|$.
2. Check if $ELen \geq 2 \cdot HLen + 1$; if not, then fail.
3. Parse $E$ as $E = t \,\|\, s$, where $|t| = HLen$ and $|s| = ELen - HLen$.
4. Set $r = KDF(s, HLen) \oplus t$.
5. Set $x = KDF(r, ELen - HLen) \oplus s$.
6. Test that $x$ is of the form $x = Hash.eval(L) \,\|\, pad \,\|\, M$, where *pad* is a byte string consisting of zero or more 0-bytes, followed by a 1-byte. If not, then fail.
7. Output $M$.

## 18.3 *RSA-OAEP*

We describe a generic RSA encryption scheme, based on an arbitrary message encoding mechanism *EME*. If one uses *OAEP-EME*, the resulting scheme is called *RSA-OAEP*.

*RSA-OAEP* is a *bounded length* public-key encryption scheme.

### 18.3.1 Key generation

The public key consists of an RSA modulus $n$ that is the product of two large primes, and an exponent $e$, where $\gcd(e, \phi(n)) = 1$. It also specifies any parameters of *EME* (such as *Hash* and *KDF*, in the case of *OAEP-EME*). Let *nLen* denote the length, in bytes, of $n$.

The secret key consists of the decryption exponent $d$, where $ed \equiv 1 \bmod \phi(n)$.

### 18.3.2 Encryption

The algorithm to encrypt a message $M$, where $|M| \leq nLen - EME.Bound - 1$, with label $L$ runs as follows.

1. Set $E = EME.Encode(M, L, nLen - 1)$.
2. Set $m = OS2IP(E)$.
3. Set $c = m^e \bmod n$.
4. Output $C = I2OSP(c, nLen)$.

### 18.3.3 Decryption

The algorithm to decrypt a ciphertext $C$ with label $L$ runs as follows.

1. If $|C| \neq nLen$, then fail.
2. Let $c = OS2IP(C)$.
3. Check that $c \leq n - 1$; if not, then fail.
4. Set $m = c^d \bmod n$.
5. Set $E = I2OSP(m, nLen - 1)$; note that this step may fail if $m$ is too large.
6. Set $M = EME.Decode(E, L)$; note that this step may fail.

An implementation should take care not to reveal which of steps 5 or 6 fail. Such information could take the form of distinct error codes, or of timing information. In particular, it is recommended that both steps 5 and 6 should be performed, even if step 5 fails. If such precautions are not taken, an implementation may be vulnerable to Manger's attack [Man01].

### 18.4 Defects of *RSA-OAEP*

*RSA-OAEP* suffers from two defects.

The first is a security defect. It was a widely held belief that the general OAEP construction was secure against adaptive chosen ciphertext attack, assuming the underlying trapdoor permutation was one-way. This belief is based on a supposed random-oracle proof in [BR94]. This of course would imply the security of *RSA-OAEP* in the random oracle model, assuming that RSA is one-way. However, it was recently shown in [Sho01] that the proof of security of the general OAEP construction was invalid, and further, the general construction can not be proven secure using standard proof techniques.

This result by itself does not imply that *RSA-OAEP* is insecure; it simply invalidates the original justification of its security. In fact, in [Sho01], it is shown that *RSA-OAEP* with $e = 3$ is secure (in the random oracle model). This result is extended by [FOPS01] to arbitrary $e$. It should be noted however, that the security reduction is much less efficient in [FOPS01] than that proposed in [BR94] for OAEP.

The fact that *RSA-OAEP* can be proved secure is essentially an accident. The proofs of security exploit particular algebraic properties of the RSA function.

In [Sho01], a slight variant of OAEP is presented, called OAEP+. A detailed proof of security is given, on the general assumption of a trapdoor one-way permutation. Moreover, the security reduction is much more efficient than that of [FOPS01] or even [BR94].

Another defect of *RSA-OAEP* is that it only encrypts messages of a bounded length. Because of this, *RSA-OAEP* is really only useful as a key encapsulation mechanism (see §3), and it is left to application engineers to implement a "digital envelope" for encrypting longer messages. See §2.1.2 for a discussion about why we believe that this standard should provide a complete solution to the "digital envelope" problem, rather than just a partial solution. Also, using *RSA-OAEP* for nothing more than key encapsulation completely wastes one of the main feature of OAEP, namely, its very good "message expansion" rate. Indeed, if all one wants to do with RSA is encapsulate a key, then one is better served using the *RSA-KEM* scheme in §20, as that method is both simpler and quantitatively more secure.

Because of these two defects, we propose that the new ISO standard contain a variation of *RSA-OAEP+* that offers both a higher level of security than *RSA-OAEP*, while at the same time introduces a standard for encrypting messages of arbitrary length using RSA.

## 19 *RSA-OAEP+*

In this section, we propose a new encryption scheme, called *RSA-OAEP+*. It has better provable security properties than *RSA-OAEP*, and also provides a secure mechanism for encrypting messages of arbitrary length.

### 19.1 Extended message encoding functions

To facilitate encryption of arbitrary length messages, we extend the notion of a message encoding scheme.

In general, an *extended* message encoding scheme *XEME* specifies two algorithms:

- *XEME.Encode*$(M, L, ELen, KeyLen)$ takes as input a message $M$, a label $L$, an encoding output length *ELen*, and a key output length *KeyLen*. Here, $M$ and $L$ are byte strings whose lengths are bounded, as described below. It outputs a pair $(E, K)$ of byte strings with $|E| = \mathit{ELen}$ and $|K| = \mathit{KeyLen}$.

- $\mathit{XEME.Decode}(E, L, \mathit{KeyLen})$ takes as input a byte string $E$ and a label $L$. It attempts to find a message $M$ and a key $K$ such that $\mathit{EME.Encode}(M, L, |E|, \mathit{KeyLen}) = (E, K)$. It returns the pair $(M, K)$ if it exists, and otherwise fails.

In addition to this, the mechanism should specify a bound $\mathit{XEME.Bound}$ such that when $\mathit{XEME.Encode}(M, L, \mathit{ELen}, \mathit{KeyLen})$ is invoked, the condition $|M| \leq \mathit{ELen} - \mathit{XEME.Bound}$ should hold; if not, the encoding algorithm fails. Additionally, the encoding algorithm may also fail if $|L|$ or $\mathit{KeyLen}$ exceed some (very large) implementation-defined bound.

The algorithm $\mathit{XEME.Encode}$ will in general be probabilistic, so that the same message can be encoded in a number of ways.

## 19.2 *OAEP+XEME*

We now describe the extended message encoding scheme $\mathit{OAEP{+}XEME}$.

The scheme is parameterized by a key derivation function $\mathit{KDF}$ (see §12) and an integer $\mathit{MaskLen} \geq 1$. Any of the functions described in §12 are suitable.

The quantity $\mathit{OAEP{+}XEME.Bound}$ is defined as

$$\mathit{OAEP{+}XEME.Bound} = 2 \cdot \mathit{MaskLen} + 1.$$

Let $(\mathit{I0}, \mathit{I1}, \ldots)$ denote the values $(\mathit{I2OSP}(0, 4), \mathit{I2OSP}(1, 4), \ldots)$.

### 19.2.1 Encoding function

The algorithm $\mathit{OAEP{+}XEME.Encode}(M, L, \mathit{ELen}, \mathit{KeyLen})$ runs as follows.

1. Check that $|M| <= \mathit{ELen} - 2 \cdot \mathit{MaskLen} - 1$; if not, then fail.

2. Generate a random byte string $r$ of length $\mathit{MaskLen}$.

3. Let $\mathit{pad}$ be the byte string of length $\mathit{ELen} - |M| - 2 \cdot \mathit{MaskLen}$ consisting of a sequence of 0-bytes, followed by a single 1-byte.

4. Set $x = \mathit{pad} \,\|\, M$.

5. Set
$$\mathit{check} = \mathit{KDF}(\mathit{I0} \,\|\, r \,\|\, x \,\|\, \mathit{I2OSP}(\mathit{KeyLen}, 4) \,\|\, L, \mathit{MaskLen}).$$

6. Set
$$x' = \mathit{KDF}(\mathit{I1} \,\|\, r, \mathit{ELen} - 2 \cdot \mathit{MaskLen}) \oplus x.$$

7. Set
$$s = \mathit{check} \,\|\, x'.$$

8. Set
$$t = \mathit{KDF}(\mathit{I2} \,\|\, s, \mathit{MaskLen}) \oplus r.$$

9. Output
$$E = t \,\|\, s$$
and
$$K = \mathit{KDF}(\mathit{I3} \,\|\, r, \mathit{KeyLen}).$$

### 19.2.2 Decoding function

The algorithm $\mathit{OAEP{+}XEME.Decode}(E, L)$ runs as follows.

1. Let $\mathit{ELen} = |E|$.
2. Check if $\mathit{ELen} \geq 2 \cdot \mathit{MaskLen} + 1$; if not, then fail.
3. Parse $E$ as $E = t \,\|\, s$, where $|t| = \mathit{MaskLen}$ and $|s| = \mathit{ELen} - \mathit{MaskLen}$.
4. Set
$$r = \mathit{KDF}(\mathit{I2} \,\|\, s, \mathit{MaskLen}) \oplus t.$$
5. Parse $s$ as $\mathit{check} \,\|\, x'$, where $|\mathit{check}| = \mathit{MaskLen}$ and $|x'| = \mathit{ELen} - 2 \cdot \mathit{MaskLen}$.
6. Set
$$x = \mathit{KDF}(\mathit{I1} \,\|\, r, \mathit{ELen} - 2 \cdot \mathit{MaskLen}) \oplus x'.$$
7. Test if $x$ is of the form $x = \mathit{pad} \,\|\, M$, where $\mathit{pad}$ is a byte string consisting of zero or more 0-bytes, followed by a 1-byte; if not, then fail.
8. Test if
$$\mathit{check} = \mathit{KDF}(\mathit{I0} \,\|\, r \,\|\, x \,\|\, \mathit{I2OSP}(\mathit{KeyLen}, 4) \,\|\, L, \mathit{MaskLen}).$$
If not, then fail.
9. Output $M$ and
$$K = \mathit{KDF}(\mathit{I3} \,\|\, r, \mathit{KeyLen}).$$

This encoding scheme is very similar to that of [Sho01]. Besides a few inconsequential formatting changes, this scheme deals with a label $L$ and produces a key $K$ of length $\mathit{KeyLen}$. The scheme in [Sho01] does not deal with labels or key outputs at all. Notice that both $\mathit{KeyLen}$ and $L$ are hashed into the value $\mathit{check}$ — this is important for the security of the scheme.

In general, we have kept the changes between *OAEP-EME* and *OAEP+XEME* minimal. But since some changes were anyway necessary, we took the liberty to propose a couple of further changes.

The main change is that we use the function *KDF* in several places, and we insert the strings $\mathit{I0}, \mathit{I1}$, etc., into the different invocations of *KDF*. This is done so that these can be more properly modeled as *independent* random oracles, as required in the proof of security.

## 19.3 *RSA-OAEP+*

We describe a generic *extended* RSA encryption scheme that uses an arbitrary *extended* message encoding scheme *XEME*. If the *OAEP+XEME* encoding scheme is used, the resulting encryption scheme is called *RSA-OAEP+*. We call this an extended RSA encryption scheme, since it handles messages of arbitrary length.

This scheme also makes use of a data encapsulation mechanism *DEM* (see §4); however, we do not require that *DEM* supports any labels.

### 19.3.1 Key generation

Just as for *RSA-OAEP*, the public key consists of an RSA modulus $n$ that is the product of two large primes, and an exponent $e$, where $\gcd(e, \phi(n)) = 1$. It also specifies any parameters of *XEME*. Let *nLen* denote the length, in bytes, of $n$.

The secret key consists of the decryption exponent $d$, where $ed \equiv 1 \bmod \phi(n)$.

### 19.3.2 Encrypting short messages

To encrypt a message $M$ with label $L$, where $|M| \leq \mathit{nLen} - \mathit{XEME.Bound} - 1$, one does the following.

1. Set $(E, K) = \mathit{XEME.Encode}(M, L, \mathit{nLen} - 1, 0)$; note that $K$ is the empty string.
2. Set $m = \mathit{OS2IP}(E)$.
3. Set $c = m^e \bmod n$.
4. Output $C = \mathit{I2OSP}(c, \mathit{nLen})$.

### 19.3.3 Decrypting short messages

To decrypt a ciphertext $C$ with label $L$, where $|C| \leq \mathit{nLen}$, one does the following.

1. If $|C| < \mathit{nLen}$, then fail.
2. Let $c = \mathit{OS2IP}(C)$.
3. Check that $c \leq n - 1$; if not, then fail.
4. Set $m = c^d \bmod n$.
5. Set $E = \mathit{I2OSP}(m, \mathit{nLen} - 1)$; note that this step may fail if $m$ is too large.
6. Set $(M, K) = \mathit{XEME.Decode}(E, L, 0)$; note that this step may fail, and also that $K$ is the empty string.
7. Output $M$.

As in the the case of *RSA-OAEP*, an implementation should reveal no information that would reveal to an adversary which of steps 5 or 6 fail.

### 19.3.4 Encrypting long messages

To encrypt a message $M$ with label $L$, where $|M| > \mathit{nLen} - \mathit{XEME.Bound} - 1$, one does the following.

1. Let $M = M_0 \parallel M_1$, where $|M_0| = \mathit{nLen} - \mathit{XEME.Bound} - 1$.
2. Set $(E, K) = \mathit{XEME.Encode}(M_0, L, \mathit{nLen} - 1, \mathit{DEM.KeyLen})$.
3. Set $m = \mathit{OS2IP}(E)$.
4. Set $c = m^e \bmod n$.

5. Set $C_0 = \mathit{I2OSP}(c, \mathit{nLen})$.
6. Encrypt $M_1$ under the key $K$ using $\mathit{DEM}$, and let $C_1$ be the resulting ciphertext.
7. Output the ciphertext $C = C_0 \,\|\, C_1$.

### 19.3.5 Decrypting long messages

To decrypt a ciphertext $C$ with label $L$, where $|C| > \mathit{nLen}$, one does the following.

1. Parse $C$ as $C = C_0 \,\|\, C_1$, where $|C_0| = \mathit{nLen}$.
2. Let $c = \mathit{OS2IP}(C_0)$.
3. Check that $c \leq n - 1$; if not, then fail.
4. Set $m = c^d \bmod n$.
5. Set $E = \mathit{I2OSP}(m, \mathit{nLen} - 1)$; note that this step may fail if $m$ is too large.
6. Set $(M_0, K) = \mathit{XEME.Decode}(E, L, \mathit{DEM.KeyLen})$. Note that this step may fail.
7. Test if $|M_0| = \mathit{nLen} - \mathit{XEME.Bound} - 1$; if not, then fail.
8. Decrypt $C_1$ under the key $K$ using $\mathit{DEM}$, and let $M_1$ be the resulting message.
9. Output $M = M_0 \,\|\, M_1$.

As in the the case of *RSA-OAEP*, an implementation should reveal no information that would reveal to an adversary which of steps 5 or 6 fail.

## 19.4 Security considerations

It is straightforward to adapt the proof of security in [Sho01] to show that this scheme is secure in the random oracle model against adaptive chosen ciphertext attack, assuming the RSA inversion problem is hard.

That proof implies that for any adversary $A$, its advantage in breaking the cryptosystem *RSA-OAEP+* is bounded by

$$\begin{aligned} \mathit{Advantage}_{\mathit{RSA\text{-}OAEP+}}(A) \; = O( \; & \mathit{Advantage}_{\mathit{RSA}}(A_1) \; + \\ & \mathit{Advantage}_{\mathit{DEM}}(A_2, l_1) \; + \\ & q_D \cdot q_{\mathit{KDF}} \cdot 2^{-\mathit{MaskLen}} \; ) \end{aligned}$$

Here,

- $A_1$ is an algorithm that runs in time roughly equivalent to that of $A$, plus $O(q_{\mathit{KDF}}^2)$ applications of the RSA function,
- $A_2, A_3$ are adversaries whose running times are about the same as $A$,
- $\mathit{Advantage}_{\mathit{RSA}}(A)$ denotes the success probability of an algorithm $A$ has in solving a random instance of the RSA inversion problem,
- $q_D$ is a bound on the number of decryption oracle queries made by $A$,

- $q_{KDF}$ is a bound on the number of random oracle queries made by $A$,
- $l_1$ is a bound on the length of the target message, and
- $l_2$ is a bound on the length of ciphertexts submitted to the decryption oracle.

Note that this security reduction is actually somewhat more efficient than the original (and incorrect) security reduction for *RSA-OAEP* in [BR94]. It is also far more efficient than the security reduction in [FOPS01]. In that reduction, the algorithm $A'$ for inverting RSA is somewhat slower than that of *RSA-OAEP+*, but worse, if the advantage of $A$ is $\epsilon$, then the success probability of $A'$ is about $\epsilon^2$.

Even though the security reduction for *RSA-OAEP+* is tighter than that for *RSA-OAEP*, we should perhaps point out that because of the term $O(q_{KDF}^2)$ in the running time of the RSA inversion algorithm, this reduction actually says very little about the security of, say, 1024-bit RSA. This is because one can (most likely) factor 1024-bit numbers in less time than that required by the implied RSA inversion algorithm. However, as pointed out in [Sho01], for exponent $e = 3$, there is a much more efficient security reduction whose running time is linear in $q_{KDF}$. Is this a reason recommend the use of $e = 3$? Perhaps. Alternatively, one can use the *RSA-KEM* scheme (see §20).

Of course, if the security reduction for *RSA-OAEP+* implies very little about concrete security, the security reduction for *RSA-OAEP* in [FOPS01] says even less.

# 20 *RSA-KEM*

We also suggest for possible inclusion in the ISO standard the following very simple version of RSA. It is based on the ideas in [BR93].

The scheme we present is a key encapsulation mechanism (see §3), called *RSA-KEM*, which can be turned into an encryption scheme as described in §5.

The main advantages of this scheme are its simplicity and the fact that it yields a much more efficient (and hence meaningful) security reduction compared to that for OAEP or OAEP+. The disadvantage is that ciphertexts are a little bit larger.

## 20.1 Key generation

Just as for *RSA-OAEP*, the public key consists of an RSA modulus $n$ that is the product of two large primes, and an exponent $e$, where $\gcd(e, \phi(n)) = 1$. It also specifies a key derivation function *KDF* (see §12). Let *nLen* denote the length, in bytes, of $n$.

The secret key consists of the decryption exponent $d$, where $ed \equiv 1 \bmod \phi(n)$.

## 20.2 Encryption

Recall that *RSA-KEM* is a key encapsulation mechanism, and so the goal of the encryption algorithm is simply to produce a pseudo-random key $K$ of length $KeyLen = RSA\text{-}KEM.OutputKeyLen$ and a ciphertext $C_0$ that encrypts $K$.

The encryption algorithm runs as follows.

1. Generate a random number $r \in \{0, \ldots, n-1\}$.
2. Compute $y = r^e \bmod n$.
3. Compute $K = KDF(I2OSP(r, nLen), KeyLen)$.

4. Compute $C_0 = \mathit{I2OSP}(y, nLen)$.
5. Output the ciphertext $C_0$ and the key $K$.

### 20.3 Decryption

Given a ciphertext $C_0$, decryption runs as follows.

1. Check that $|C_0| = nLen$; if not, then fail.
2. Set $y = \mathit{OS2IP}(C_0)$.
3. Check that $y < n$; if not, then fail.
4. Compute $r = y^d \bmod n$.
5. Compute $K = \mathit{KDF}(\mathit{I2OSP}(r, nLen), KeyLen)$.
6. Output the key $K$.

### 20.4 Security considerations

The security of *RSA-KEM* can be analyzed in the random oracle model in a manner very similar to that in [BR93], where we model the invocation of *KDF* as a random oracle query. It is easy to show that

$$Advantage_{RSA\text{-}KEM}(A) \leq Advantage_{RSA}(A') + nBound/q_D, \tag{8}$$

where

- $A'$ is an algorithm for solving a random instance of the RSA problem that runs in time roughly the same as that of $A$; more precisely, the running time is that of $A$, plus the time to perform $q_{KDF}$ exponentiations modulo $n$, where $q_{KDF}$ is a bound on the number of random oracle queries made by $A$;
- $q_D$ is a bound on the number of decryption oracle queries made by $A$;
- *nBound* is an *lower bound* on $n$.

We sketch a proof of this. Let $\mathbf{G}_0$ be the original attack game played by adversary $A$, and let $S_0$ be the event that $A$ correctly guesses the hidden bit $b$ in game $\mathbf{G}_0$. Let $H$ denote the random oracle mapping elements of $\mathbf{Z}_n$ to bit strings of length *KeyLen*. Let $y^* \in \mathbf{Z}_n$ denote the target ciphertext, and let $r^* = (y^*)^{1/e} \in \mathbf{Z}_n$.

We next define a game $\mathbf{G}_1$ that is the same as game $\mathbf{G}_0$, except that if the target ciphertext $y^*$ was submitted to the decryption oracle prior to the invocation of the encryption oracle, then the game is halted. Let $S_1$ be the event in game $\mathbf{G}_1$ corresponding to the event $S_0$.

Let $F_1$ be the event that game $\mathbf{G}_1$ is halted as above. Clearly, $\Pr[F_1] \leq nBound/q_D$, and since games $\mathbf{G}_0$ and $\mathbf{G}_1$ proceed identically until $F_1$ occurs, it follows by Lemma 1 that $|\Pr[S_0] - \Pr[S_1]| \leq nBound/q_D$.

We next define a game $\mathbf{G}_2$ that is the same as $\mathbf{G}_1$, except that (1) the target ciphertext is generated at the beginning of the game, and (2) if the adversary ever queries $H$ at $r^*$, we halt the game. Let $S_2$ be the event in game $\mathbf{G}_2$ corresponding to the event $S_0$.

It is clear by construction that $\Pr[S_2] = 1/2$, since the key $H(r^*)$ is independent of everything else that is accessible to the adversary in game $\mathbf{G}_2$, either directly or indirectly. Indeed, only the encryption oracle evaluates $H$ at $r^*$ in this game.

Let $F_2$ be the event that game $\mathbf{G}_2$ is halted as above. It is clear that both games $\mathbf{G}_1$ and $\mathbf{G}_2$ proceed identically until $F_2$ occurs, and so by Lemma 1, we have $|\Pr[S_1] - \Pr[S_2]| \leq \Pr[F_2]$. Thus, it suffices to bound $\Pr[F_2]$.

We claim that

$$\Pr[F_2] \leq \mathit{Advantage}_{RSA}(A')$$

for an algorithm $A'$ that runs in time bounded as described above. The inequality (8) will follow immediately.

Algorithm $A'$ runs as follows. It takes as input a random RSA modulus $n$, an RSA exponent $e$, and a random element $y^* \in \mathbf{Z}_n$. It creates a public key using $N$ and $e$, and then lets adversary $A$ run in game $\mathbf{G}_2$.

When adversary $A$ invokes the encryption oracle, algorithm $A'$ responds to $A$ with the pair $(K^*, y^*)$, where $K^*$ is a random bit string of length *KeyLen*, and $y^*$ is the above-mentioned input to $A$.

Algorithm $A'$ simulates the random oracle $H$ as well as the decryption oracle, as follows. For every input $r \in \mathbf{Z}_n$ to the random oracle, $A'$ computes $y = r^e \in \mathbf{Z}_n$, and places the triple consisting of $r$, $y$, and the random value $K = H(r)$ in a table; however, if $y = y^*$, algorithm $A'$ instead outputs $r$ and halts. When the adversary $A$ submits a ciphertext $y \in \mathbf{Z}_n$ to the decryption oracle, algorithm $A'$ looks up the value $y$ in the above table to determine if the random oracle has been evaluated at $r = y^{1/e} \in \mathbf{Z}_n$. If so, algorithm $A'$ responds to the decryption oracle invocation with the value $K = H(r)$ stored in the table. Otherwise, algorithm $A'$ generates a fresh random key $K$, and places the pair $(y, K)$ in a second table; moreover, if in the future the adversary $A$ should evaluate the random oracle at a point $r \in \mathbf{Z}_n$ such that $r^e = y$, then the key $K$ generated above will be used for the value of $H(r)$.

It is clear that algorithm $A'$ perfectly simulates the view of $A$, and that $A'$ outputs a solution to the given instance of the RSA problem with probability equal to $\Pr[F_2]$.

That completes the proof of security.

Quantitatively, it is clear that *RSA-KEM* provides a much better security reduction than *RSA-OAEP+* (or *RSA-OAEP*). This advantage becomes even more pronounced when one analyzes the security of *many* messages encrypted under a single public key (as formally modeled in [BBM00]). In this setting, one can exploit the well-known random self-reducibility property of the RSA inversion problem to easily show that the security of *RSA-KEM* key encapsulation mechanism does not degrade at all as the number of ciphertexts increases. Note that this argument will be valid only if the number $r$ in the encryption algorithm for Simple RSA is chosen uniformly modulo $n$, or at least with a distribution that is computationally indistinguishable from the uniform distribution.

For *RSA-OAEP+*, the security degrades linearly with the number of ciphertexts, since one cannot use the random self-reducibility property, and must instead use a "hybrid argument." The reason the random self-reducibility property cannot be used is that in *RSA-OAEP+* (like *RSA-OAEP*) the ciphertext is not uniformly distributed modulo $n$.

We also mention that *RSA-KEM* does not appear to be as "fragile" as either *RSA-OAEP* or *RSA-OAEP+*, in the sense that there appears to be no possible attacks on an implementation, such as those in [Man01].

## 21 Further actions

In this section, we summarize the next steps that must be taken to fully develop a standard.

First, there are still a few gaps in the specifications that need to be filled in:

**choice of groups for Diffie-Hellman schemes:** We have yet to specify precisely what groups are allowed. One possibility is to simply adopt "wholesale" the choices allowable in IEEE P1363.

**encoding of group elements:** We have yet to fully specify how group elements are to be encoded as byte strings. Again, one possibility is simply to adopt "wholesale" the encoding schemes in IEEE P1363.

**choice of MAC's, SKE's, and KDF's:** These choices are also yet to be specified. Again, one possibility is to simply adopt "wholesale" the choices allowable in IEEE P1363. However, the recommendations for *KDF3* and *KDF4* in the present proposal should be given some consideration. Also, for SKE's, we might want to consider other choices than those available in IEEE P1363a, e.g., a "counter mode" use of a block cipher; further, we may wish to "harmonize" the notion of an SKE as used here with the symmetric-key encryption part of the larger ISO standard.

Second, a decision must be made as to whether this standard should require or recommend a particular encoding for public keys, e.g., as ASN.1 encoded structures. An argument for doing this is that it could potentially greatly enhance adoption of the schemes in this standard.

Third, a decision must be made as to whether the schemes proposed here are the ones we really want. There seemed to be some consensus at the *ad hoc* meeting in Santa Barbara that the selection proposed here was acceptable, with the understanding that the decision to include the schemes *RSA-OAEP+* and *RSA-KEM*, which were proposed by the editor, should be taken later after further scrutiny and discussion. Also, a decision needs to be made as to whether all of the restrictions on *ECIES* discussed in §15.6 are appropriate, or whether some of these should be relaxed.

Fourth, a full specification for *EPOC-2*, hopefully compatible with IEEE P1363a, needs to be developed for inclusion in a working draft.

Fifth, this proposal must be converted to a proper draft standard. This entails separating out normative content (specifications) from informational content (rationale, security considerations, etc.), putting the latter in an annex. The intent is that the standard should contain almost all of the informational content that is in this proposal.

## References


[ABR98] M. Abdalla, M. Bellare, and P. Rogaway. DHAES: an encryption scheme based on the Diffie-Hellma problem. Submission to IEEE P1363, 1998.

[BBM00] M. Bellare, A. Boldyreva, and S. Micali. Public-key encryption in a multi-user setting: security proofs and improvements. In *Advances in Cryptology–Eurocrypt 2000*, 2000.

[BDPR98] 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*, pages 26–45, 1998.

[BLK00] J. Baek, B. Lee, and K. Kim. Secure length-saving ElGamal encryption under the computational Diffie-Hellman assumption. In *Proc. 5th Australian Conference on Information, Security, and Privacy*, 2000.

[Bon98] D. Boneh. The Decision Diffie-Hellman Problem. In *Ants-III*, pages 48–63, 1998. Springer LNCS 1423.

[BR93] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *First ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[BR94] M. Bellare and P. Rogaway. Optimal asymmetric encryption. In *Advances in Cryptology—Eurocrypt '94*, pages 92–111, 1994.

[CGH98] R. Canetti, O. Goldreich, and S. Halevi. The random oracle model, revisted. In *30th Annual ACM Symposium on Theory of Computing*, 1998.

[CS98] R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Advances in Cryptology–Crypto '98*, pages 13–25, 1998.

[DDN91] D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. In *23rd Annual ACM Symposium on Theory of Computing*, pages 542–552, 1991.

[DDN98] D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography, 1998. Manuscript (updated, full length version of STOC paper).

[FO99] E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *Advances in Cryptology–Crypto '99*, pages 537–554, 1999.

[FOPS01] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA-OAEP is secure under the RSA assumption. In *Advances in Cryptology–Crypto 2001*, 2001.

[JM96] D. Johnson and S. Matya. Asymmetric encryption: evolution and enhancements. *Cryptobytes*, 2(1), 1996. `http://www.rsasecurity.com/rsalabs`.

[JN01] A. Joux and K. Nguyen. Separating Decision Diffie-Hellman from Diffie-Hellman in cryptographic groups. Cryptology ePrint Archive, Report 2001/003, 2001. `http://eprint.iacr.org`.

[Luc00] S. Lucks. The sum of PRPs is a secure PRF. In *Advances in Cryptology–Eurocrypt 2000*, 2000.

[Man01] J. Manger. A chosen ciphertext attack on RSA Optimal Asymmetric Encryption Padding (OAEP) as standardized in PKCS # 1 v2.0. In *Advances in Cryptology–Crypto 2001*, pages 230–238, 2001.

[NR97] M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *38th Annual Symposium on Foundations of Computer Science*, 1997.

[OP01] T. Okamoto and D. Pointcheval. The gap-problems: a new class of problems for the security of cryptographic schemes. In *Proc. 2001 International Workshop on Practice and Theory in Public Key Cryptography (PKC 2001)*, 2001.

[RS91] C. Rackoff and D. Simon. Noninteractive zero-knowledge proof of knowledge and chosen ciphertext attack. In *Advances in Cryptology–Crypto '91*, pages 433–444, 1991.

[Sho97] V. Shoup. Lower bounds for discrete logarithms and related problems. In *Advances in Cryptology–Eurocrypt '97*, 1997.

[Sho00] V. Shoup. Using hash functions as a hedge against chosen ciphertext attack. In *Advances in Cryptology–Eurocrypt 2000*, 2000.

[Sho01] V. Shoup. OAEP reconsidered. In *Advances in Cryptology–Crypto 2001*, 2001.

[Sta96] M. Stadler. Publicly verifiable secret sharing. In *Advances in Cryptology–Eurocrypt '96*, pages 190–199, 1996.