

Australian/New Zealand Standard™

**Information technology—Security
techniques—Encryption algorithms**

Part 2: Asymmetric ciphers



AS/NZS ISO/IEC 18033.2:2008

This Joint Australian/New Zealand Standard was prepared by Joint Technical Committee IT-012, Information Systems, Security and Identification Technology. It was approved on behalf of the Council of Standards Australia on 28 May 2008 and on behalf of the Council of Standards New Zealand on 31 May 2008. This Standard was published on 25 June 2008.

The following are represented on Committee IT-012:

Attorney General's Office
Australian Association of Permanent Building Societies
Australian Bankers Association
Australian Chamber of Commerce and Industry
Australian Electrical and Electronic Manufacturers Association
Certification Forum of Australia
Council of Small Business Organisations
Internet Industry Association
NSW Police
New Zealand Defence Force
Reserve Bank of Australia

Keeping Standards up-to-date

Standards are living documents which reflect progress in science, technology and systems. To maintain their currency, all Standards are periodically reviewed, and new editions are published. Between editions, amendments may be issued. Standards may also be withdrawn. It is important that readers assure themselves they are using a current Standard, which should include any amendments which may have been published since the Standard was purchased.

Detailed information about joint Australian/New Zealand Standards can be found by visiting the Standards Web Shop at www.standards.com.au or Standards New Zealand web site at www.standards.co.nz and looking up the relevant Standard in the on-line catalogue.

Alternatively, both organizations publish an annual printed Catalogue with full details of all current Standards. For more frequent listings or notification of revisions, amendments and withdrawals, Standards Australia and Standards New Zealand offer a number of update options. For information about these services, users should contact their respective national Standards organization.

We also welcome suggestions for improvement in our Standards, and especially encourage readers to notify us immediately of any apparent inaccuracies or ambiguities. Please address your comments to the Chief Executive of either Standards Australia or Standards New Zealand at the address shown on the back cover.

This Standard was issued in draft form for comment as DR 07092.

Australian/New Zealand Standard™

Information technology—Security techniques—Encryption algorithms

Part 2: Asymmetric ciphers

First published as AS/NZS ISO/IEC 18033.2:2008.

COPYRIGHT

© Standards Australia/Standards New Zealand

All rights are reserved. No part of this work may be reproduced or copied in any form or by any means, electronic or mechanical, including photocopying, without the written permission of the publisher.

Jointly published by Standards Australia, GPO Box 476, Sydney, NSW 2001 and Standards New Zealand, Private Bag 2439, Wellington 6020

ISBN 0 7337 8761 4

PREFACE

This Standard was prepared by the Joint Standards Australia/Standards New Zealand Committee IT-012, Information Systems, Security and Identification Technology.

The objective of this Standard is to provide the Australian information security programming and development community with clear guidance to the selection and implementation of appropriate encryption algorithms.

This Standard is identical with, and has been reproduced from ISO/IEC 18033-2:2006, *Information technology—Security techniques—Encryption algorithms—Part 2: Asymmetric ciphers*.

As this Standard is reproduced from an international standard, the following applies:

- (a) Its number appears on the cover and title page while the international standard number appears only on the cover.
- (b) In the source text ‘this part of ISO/IEC 18033’ should read ‘this Australian/New Zealand Standard’.
- (c) A full point substitutes for a comma when referring to a decimal marker.

This Standard is Part 2 of AS/NZS 18033, *Information technology—Security techniques—Encryption algorithms*, which, when complete, will consist of the following:

Part 1: General

Part 2: Asymmetric ciphers (this Standard)

Part 3: Block ciphers

Part 4: Stream ciphers

References to International Standards should be replaced by references to Australian or Australian/New Zealand Standards, as follows:

<i>Reference to International Standard</i>		<i>Australian/New Zealand Standard</i>	
ISO/IEC		AS ISO/IEC	
10118	Information technology—Security techniques—Hash-functions	10118	Information technology—Security techniques—Hash-functions
10118-2	Part 2: Hash-functions using an n-bit block cipher	10118.2	Part 2: Hash-functions using an n-bit block cipher
10118-3	Part 3: Dedicated hash-functions	10118.3	Part 3: Dedicated hash-functions
		AS/NZS ISO/IEC	
18033	Information technology—Security techniques—Encryption algorithms	18033	Information technology—Security techniques—Encryption algorithms
18033-3	Part 3: Block ciphers	18033.3	Part 3: Block ciphers

Any international references not listed have not been adopted as Australian or Australian/New Zealand Standards.

The terms ‘normative’ and ‘informative’ are used to define the application of the annex to which they apply. A normative annex is an integral part of a standard, whereas an informative annex is only for information and guidance.

CONTENTS

	<i>Page</i>
1 Scope	1
2 Normative references	1
3 Definitions	2
4 Symbols and notation	7
5 Mathematical conventions	8
5.1 Functions and algorithms	8
5.2 Bit strings and octet strings	9
5.3 Finite Fields	10
5.4 Elliptic curves	12
6 Cryptographic transformations	14
6.1 Cryptographic hash functions	14
6.2 Key derivation functions	15
6.3 MAC algorithms	16
6.4 Block ciphers	16
6.5 Symmetric ciphers	17
7 Asymmetric ciphers	19
7.1 Plaintext length	20
7.2 The use of labels	21
7.3 Ciphertext format	21
7.4 Encryption options	21
7.5 Method of operation of an asymmetric cipher	22
7.6 Allowable asymmetric ciphers	22
8 Generic hybrid ciphers	22
8.1 Key encapsulation mechanisms	23
8.2 Data encapsulation mechanisms	24
8.3 <i>HC</i>	25
9 Constructions of data encapsulation mechanisms	26
9.1 <i>DEM1</i>	26
9.2 <i>DEM2</i>	27
9.3 <i>DEM3</i>	28
10 ElGamal-based key encapsulation mechanisms	30
10.1 Concrete groups	30
10.2 <i>ECIES-KEM</i>	32
10.3 <i>PSEC-KEM</i>	34
10.4 <i>ACE-KEM</i>	36
11 RSA-based asymmetric ciphers and key encapsulation mechanisms	39
11.1 RSA key generation algorithms	39
11.2 RSA Transform	40
11.3 RSA encoding mechanisms	40
11.4 <i>RS_AES</i>	42
11.5 <i>RSA-KEM</i>	44
12 Ciphers based on modular squaring	45

12.1	HIME key generation algorithms	45
12.2	HIME encoding mechanisms	46
12.3	<i>HIME(R)</i>	48
Annex A (normative)	ASN.1 syntax for object identifiers	51
Annex B (informative)	Security considerations	61
B.1	MAC algorithms	61
B.2	Block ciphers	62
B.3	Symmetric ciphers	62
B.4	Asymmetric ciphers	63
B.5	Key encapsulation mechanisms	65
B.6	Data encapsulation mechanisms	66
B.7	Security of <i>HC</i>	68
B.8	Intractability assumptions related to concrete groups	68
B.9	Security of <i>ECIES-KEM</i>	69
B.10	Security of <i>PSEC-KEM</i>	71
B.11	Security of <i>ACE-KEM</i>	71
B.12	The RSA inversion problem	72
B.13	Security of <i>RSAES</i>	73
B.14	Security of <i>RSA-KEM</i>	73
B.15	Security of <i>HIME(R)</i>	74
Annex C (informative)	Test vectors	75
C.1	Test vectors for <i>DEM1</i>	75
C.2	Test vectors for <i>ECIES-KEM</i>	76
C.3	Test vectors for <i>PSEC-KEM</i>	83
C.4	Test vectors for <i>ACE-KEM</i>	91
C.5	Test vectors for <i>RSAES</i>	100
C.6	Test vectors for <i>RSA-KEM</i>	105
C.7	Test vectors for <i>HC</i>	109
C.8	Test vectors for <i>HIME(R)</i>	112
Bibliography		123

AUSTRALIAN/NEW ZEALAND STANDARD

Information technology — Security techniques — Encryption algorithms —**Part 2:
Asymmetric ciphers****1 Scope**

This part of ISO/IEC 18033 specifies several asymmetric ciphers. These specifications prescribe the functional interfaces and correct methods of use of such ciphers in general, as well as the precise functionality and cipher text format for several specific asymmetric ciphers (although conforming systems may choose to use alternative formats for storing and transmitting cipher-texts).

A normative annex (Annex A) gives ASN.1 syntax for object identifiers, public keys, and parameter structures to be associated with the algorithms specified in this part of ISO/IEC 18033.

However, these specifications do not prescribe protocols for reliably obtaining a public key, for proof of possession of a private key, or for validation of either public or private keys; see ISO/IEC 11770-3 for guidance on such key management issues.

The asymmetric ciphers that are specified in this part of ISO/IEC 18033 are indicated in Clause 7.6.

NOTE Briefly, the asymmetric ciphers are:

- ECIES-HC; PSEC-HC; ACE-HC: generic hybrid ciphers based on ElGamal encryption;
- RSA-HC: a generic hybrid cipher based on the RSA transform;
- RSAES: the OAEP padding scheme applied to the RSA transform;
- HIME(R): a scheme based on the hardness of factoring.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 9797-1:1999, *Information technology — Security techniques — Message Authentication Codes (MACs) — Part 1: Mechanisms using a block cipher*

ISO/IEC 9797-2:2002, *Information technology — Security techniques — Message Authentication Codes (MACs) — Part 2: Mechanisms using a dedicated hash-function*

ISO/IEC 10118-2:2000, *Information technology — Security techniques — Hash-functions — Part 2: Hash-functions using an n-bit block cipher*

ISO/IEC 10118-3:2004, *Information technology — Security techniques — Hash-functions — Part 3: Dedicated hash-functions*

ISO/IEC 18033-3:2005, *Information technology — Security techniques — Encryption algorithms — Part 3: Block ciphers*

3 Definitions

For the purposes of this document, the following terms and definitions apply.

NOTE Where appropriate, forward references are given to clauses which contain more detailed definitions and/or further elaboration.

3.1
asymmetric cipher
system based on asymmetric cryptographic techniques whose public transformation is used for encryption and whose private transformation is used for decryption

[ISO/IEC 18033-1]

NOTE See Clause 7.

3.2
asymmetric cryptographic technique
cryptographic technique that uses two related transformations, a public transformation (defined by the public key) and a private transformation (defined by the private key). The two transformations have the property that, given the public transformation, it is computationally infeasible to derive the private transformation.

[ISO/IEC 11770-1:1996]

3.3
asymmetric key pair
pair of related keys, a public key and a private key, where the private key defines the private transformation and the public key defines the public transformation

[ISO/IEC 9798-1:1997]

NOTE See Clauses 7, 8.1.

3.4
bit
one of the two symbols '0' or '1'

NOTE See Clause 5.2.1.

3.5
bit string
sequence of bits

NOTE See Clause 5.2.1.

3.6
block
string of bits of a defined length

[ISO/IEC 18033-1]

NOTE In this part of ISO/IEC 18033, a block will be restricted to be an octet string (interpreted in a natural way as a bit string).

3.7
block cipher
symmetric cipher with the property that the encryption algorithm operates on a block of plain-text, i.e., a string of bits of a defined length, to yield a block of cipher text

[ISO/IEC 18033-1]

NOTE See Clause 6.4.

NOTE In this part of ISO/IEC 18033, plaintext/cipher text blocks will be restricted to be octet strings (interpreted in a natural way as bit strings).

3.8**cipher**

cryptographic technique used to protect the confidentiality of data, and which consists of three component processes: an encryption algorithm, a decryption algorithm, and a method for generating keys
[ISO/IEC 18033-1]

3.9**cipher text**

data which has been transformed to hide its information content
[ISO/IEC 10116:1997]

3.10**concrete group**

explicit description of a finite abelian group, together with algorithms for performing the group operation and for encoding and decoding group elements as octet strings

NOTE See Clause 10.1.

3.11**cryptographic hash function**

function that maps octet strings of any length to octet strings of fixed length, such that it is computationally infeasible to find correlations between inputs and outputs, and such that given one part of the output, but not the input, it is computationally infeasible to predict any bit of the remaining output. The precise security requirements depend on the application.

NOTE See Clause 6.1.

3.12**data encapsulation mechanism**

cryptographic mechanism, based on symmetric cryptographic techniques, which protects both the confidentiality and the integrity of data

NOTE See Clause 8.2.

3.13**decryption**

reversal of the corresponding encryption
[ISO/IEC 11770-1:1996]

3.14**decryption algorithm**

process which transforms ciphertext into plaintext
[ISO/IEC 18033-1]

3.15**encryption**

(reversible) transformation of data by a cryptographic algorithm to produce cipher text, i.e., to hide the information content of the data
[ISO/IEC 9797-1]

3.16**explicitly given finite field**

finite field that is represented explicitly in terms of its characteristic and a multiplication table for a basis of the field over the underlying prime field

NOTE See Clause 5.3.

3.17**encryption algorithm**

process which transforms plaintext into cipher text

[ISO/IEC 18033-1]

3.18**encryption option**

option that may be passed to the encryption algorithm of an asymmetric cipher, or of a key encapsulation mechanism, to control the formatting of the output cipher text

NOTE See Clauses 7, 8.1.

3.19**field**

mathematical notion of a field, i.e., a set of elements, together with binary operations for addition and multiplication on this set, such that the usual field axioms apply

3.20**finite abelian group**

group such that the underlying set of elements is finite, and such that the underlying binary operation is commutative

3.21**finite field**

field such that the underlying set of elements is finite

3.22**group**

mathematical notion of a group, i.e., a set of elements, together with a binary operation on this set, such that the usual group axioms apply

3.23**hybrid cipher**

asymmetric cipher that combines both asymmetric and symmetric cryptographic techniques

3.24**key**

sequence of symbols that controls the operation of a cryptographic transformation (e.g., encryption, decryption)

[ISO/IEC 11770-1:1996]

3.25**key derivation function**

a function that maps octet strings of any length to octet strings of an arbitrary, specified length, such that it is computationally infeasible to find correlations between inputs and outputs, and such that given one part of the output, but not the input, it is computationally infeasible to predict any bit of the remaining output. The precise security requirements depend on the application.

NOTE See Clause 6.2.

3.26**key encapsulation mechanism**

similar to an asymmetric cipher, but the encryption algorithm takes as input a public key and generates a secret key and an encryption of this secret key

NOTE See Clause 8.1.

3.27**key generation algorithm**

method for generating asymmetric key pairs

NOTE See Clauses 7, 8.1.

3.28**label**

octet string that is input to both the encryption and decryption algorithms of an asymmetric cipher, and of a data encapsulation mechanism. A label is public information that is bound to the cipher text in a non-malleable way

NOTE See Clauses 7, 8.2.

3.29**length**

length of a string of digits or the representation of an integer

Specifically:

(1) length of a bit string is the number of bits in the string

NOTE See Clause 5.2.1.

(2) length of an octet string is the number of octets in the string

NOTE See Clause 5.2.2.

(3) bit length of a non-negative integer n is the number of bits in its binary representation, i.e., $\text{dlog}_2(n + 1)$

NOTE See Clause 5.2.4.

(4) octet length of a non-negative integer n is the number of digits in its representation base 256, i.e., $\text{dlog}_{256}(n + 1)$

NOTE See Clause 5.2.4.

3.30**message authentication code (MAC)**

string of bits which is the output of a MAC algorithm

[ISO/IEC 9797-1]

NOTE See Clause 6.3.

NOTE In this part of ISO/IEC 18033, a MAC will be restricted to be an octet string (interpreted in a natural way as a bit string).

3.31**MAC algorithm**

algorithm for computing a function which maps strings of bits and a secret key to fixed-length strings of bits, satisfying the following two properties:

- for any key and any input string, the function can be computed efficiently;
- for any fixed key, and given no prior knowledge of the key, it is computationally infeasible to compute the function value on any new input string, even given knowledge of the set of input strings and corresponding function values, where the value of the i th input string may have been chosen after observing the value of the first $i - 1$ function values.

[ISO/IEC 9797-1]

NOTE See Clause 6.3.

NOTE In this part of ISO/IEC 18033, the input and output strings of a MAC algorithm will be restricted to be octet strings (interpreted in a natural way as bit strings).

3.32**octet**

a bit string of length 8

NOTE See Clause 5.2.2.

3.33**octet string**

a sequence of octets

NOTE See Clause 5.2.2.

NOTE When appropriate, an octet string may be interpreted as a bit string, simply by concatenating all of the component octets.

3.34**plaintext**

unencrypted information

[ISO/IEC 10116:1997]

3.35**prefix free set**

a set S of bit/octet strings such that there do not exist strings $x \neq y \in S$ such that x is a prefix of y

3.36**primitive**

a function used to convert between data types

3.37**private key**

the key of an entity's asymmetric key pair which should only be used by that entity

[ISO/IEC 11770-1:1996]

NOTE See Clauses 7, 8.1.

3.38**public key**

the key of an entity's asymmetric key pair which can be made public

[ISO/IEC 11770-1:1996]

NOTE See Clauses 7, 8.1.

3.39**secret key**

key used with symmetric cryptographic techniques by a specified set of entities

[ISO/IEC 11770-3:1999]

3.40**symmetric cipher**

cipher based on symmetric cryptographic techniques that uses the same secret key for both the encryption and decryption algorithms

[ISO/IEC 18033-1]

3.41**system parameters**

choice of parameters that selects a particular cryptographic scheme or function from a family of cryptographic schemes or functions

4 Symbols and notation

For the purposes of this document, the following symbols and notation apply.

NOTE Where appropriate, forward references are given to clauses which contain more detailed definitions and/or further elaboration.

$\lfloor x \rfloor$	the largest integer less than or equal to the real number x . For example, $\lfloor 5 \rfloor = 5$, $\lfloor 5.3 \rfloor = 5$, and $\lfloor -5.3 \rfloor = -6$
$\lceil x \rceil$	the smallest integer greater than or equal to the real number x . For example, $\lceil 5 \rceil = 5$, $\lceil 5.3 \rceil = 6$, and $\lceil -5.3 \rceil = -5$
$[a..b]$	the interval of integers from a to b , including both a and b
$[a..b)$	the interval of integers from a to b , including a but not b
$ X $	if X is a finite set, then the cardinality of X ; if X is a finite abelian group or a finite field, then the cardinality of the underlying set of elements; if X is a real number, then the absolute value of X ; if X is a bit/octet string, then the length in bits/octets of the string NOTE See Clauses 5.2.1, 5.2.2.
$x \oplus y$	if x and y are bit/octet strings of the same length, the bit-wise exclusive-or (XOR) of the two strings. NOTE See Clauses 5.2.1, 5.2.2.
$\langle x_1, \dots, x_l \rangle$	if $x_1, \dots, x_l$ are bits/octets, the bit/octet string of length l consisting of the bits/octets $x_1; \dots; x_l$, in the given order NOTE See Clauses 5.2.1, 5.2.2.
$x y$	if x and y are bit/octet strings, the concatenation of the two strings x and y , resulting in the string consisting of x followed by y NOTE See Clauses 5.2.1, 5.2.2.
$\gcd(a, b)$	for integers a and b , the greatest common divisor of a and b , i.e., the largest positive integer that divides both a and b (or 0 if $a = b = 0$)
$a b$	relation between integers a and b that holds if and only if a divides b , i.e., there exists an integer c such that $b = ac$
$a \equiv b \pmod{n}$	for a non-zero integer n , a relation between integers a and b that holds if and only if a and b are congruent modulo n , i.e., $n (a - b)$
$a \bmod n$	for integer a and positive integer n , the unique integer $r \in [0..n)$ such that $r \equiv a \pmod{n}$
$a^{-1} \bmod n$	for integer a and positive integer n , such that $\gcd(a; n) = 1$, the unique integer $b \in [0..n)$ such that $ab \equiv 1 \pmod{n}$
F^*	for a field F , the multiplicative group of units of F
0_F	for a field F , the additive identity (zero element) of F
1_F	for a field F , the multiplicative identity of F
$BS2IP$	bit string to integer conversion primitive NOTE See Clause 5.2.5.

<i>EC2OSP</i>	elliptic curve to octet string conversion primitive. (See Clause 5.4.3.)
<i>FE2OSP</i>	field element to octet string conversion primitive. (See Clause 5.3.1.)
<i>FE2IP</i>	field element to integer conversion primitive. (See Clause 5.3.1.)
<i>I2BSP</i>	integer to bit string conversion primitive. (See Clause 5.2.5.)
<i>I2OSP</i>	integer to octet string conversion primitive. (See Clause 5.2.5.)
<i>OS2ECP</i>	octet string to elliptic curve conversion primitive. (See Clause 5.4.3.)
<i>OS2FEP</i>	octet string to field element conversion primitive. (See Clause 5.3.1.)
<i>OS2IP</i>	octet string to integer conversion primitive. (See Clause 5.2.5.)
<i>Oct(m)</i>	the octet whose integer value is <i>m</i> . (See Clause 5.2.4.)
$\mathcal{L}(n)$	the length in octets of an integer <i>n</i> . (See Clause 5.2.5.)

5 Mathematical conventions

This clause describes certain mathematical conventions used in this part of ISO/IEC 18033, including the representation of mathematical objects, and primitives for data type conversion.

5.1 Functions and algorithms

For ease of presentation, functions and probabilistic functions (i.e., functions whose value depends not only on the input value but also on a randomly chosen auxiliary value) are often specified in algorithmic form. Except where explicitly noted, an implementor may choose to employ any equivalent algorithm (i.e., one which yields the same function or probabilistic function). Moreover, in the case of probabilistic functions, when the algorithm describing the function indicates that a random value should be generated, an implementor shall use an appropriate random generator to generate this value (see ISO/IEC 18031 for more guidance on this issue).

In describing a function in algorithmic terms, the following convention is adopted. An algorithm either computes a value, or alternatively, it may **fail**. By convention, if an algorithm **fails**, then unless otherwise specified, another algorithm that invokes this algorithm as a sub-routine also **fails**.

NOTE Thus, **failing** is analogous to the notion of “throwing an exception” in many programming languages; however, it can also be viewed as returning a special value that is by definition distinct from all values returned by the algorithm when it does not **fail**. With this latter interpretation of **failing**, an algorithm still properly describes a function. The details of how an implementation achieves the effect of **failing** are not specified here. However, in a typical implementation, an algorithm may return an “error code” of some sort to its environment that indicates the reason for the failure. It should be noted that in some cases, for reasons of security, the implementation should take care *not* to reveal the precise cause of certain types of errors.

5.2 Bit strings and octet strings

5.2.1 Bits and bit strings

A *bit* is one of the two symbols ‘0’ or ‘1’.

A *bit string* is a sequence of bits. For bits $x_1, \dots, x_l$, $\langle x_1, \dots, x_l \rangle$ denotes the bit string of length l consisting of the bits $x_1, \dots, x_l$, in the given order.

For a bit string $x = \langle x_1, \dots, x_l \rangle$, the length l of x is denoted by $|x|$, and if $l > 0$, x_1 is called the *first* bit of x , and x_l the *last* bit of x .

For bit strings x and y , $x \parallel y$ denotes the concatenation of x and y ; that is, if $x = \langle x_1, \dots, x_l \rangle$ and $y = \langle y_1, \dots, y_m \rangle$, then $x \parallel y = \langle x_1, \dots, x_l, y_1, \dots, y_m \rangle$.

For bit strings x and y of equal length, $x \oplus y$ denotes the bit-wise exclusive-or (XOR) of x and y .

The bit string of length zero is called the *null* bit string.

NOTE No special subscripting operator is defined for bit strings. Thus, if x is a bit string, x_i does not necessarily denote any particular bit of x .

5.2.2 Octets and octet strings

An *octet* is a bit string of length 8.

An *octet string* is a sequence of octets.

For octets $x_1, \dots, x_l$, $\langle x_1, \dots, x_l \rangle$ denotes the octet string of length l consisting of the octets $x_1, \dots, x_l$, in the given order.

For an octet string $x = \langle x_1, \dots, x_l \rangle$, the length l of x is denoted by $|x|$, and if $l > 0$, x_1 is called the *first* octet of x , and x_l the *last* octet of x .

For octet strings x and y , $x \parallel y$ denotes the concatenation of x and y ; that is, if $x = \langle x_1, \dots, x_l \rangle$ and $y = \langle y_1, \dots, y_m \rangle$, then $x \parallel y = \langle x_1, \dots, x_l, y_1, \dots, y_m \rangle$.

For octet strings x and y of equal length, $x \oplus y$ denotes the bit-wise exclusive-or (XOR) of x and y .

The octet string of length zero is called the *null* octet string.

NOTE 1 No special subscripting operator is defined for octet strings. Thus, if x is an octet string, x_i does not necessarily denote any particular octet of x .

NOTE 2 Note that since an octet is a bit string of length 8, if x and y are octets, then $x \parallel y$ is a *bit* string of length 16, $\langle x \rangle$ and $\langle y \rangle$ are each *octet* strings of length 1, and $\langle x \rangle \parallel \langle y \rangle = \langle x, y \rangle$ is an *octet* string of length 2.

5.2.3 Octet string/bit string conversion

Primitives *OS2BSP* and *BS2OSP* to convert between octet strings and bit strings are defined as follows.

The function *OS2BSP*(x) takes as input an octet string $x = \langle x_1, \dots, x_l \rangle$, and outputs the bit string $y = x_1 \parallel \dots \parallel x_l$.

The function *BS2OSP*(y) takes as input a bit string y , whose length is a multiple of 8, and outputs the unique octet string x such that $y = \text{OS2BSP}(x)$.

5.2.4 Bit string/integer conversion

Primitives *BS2IP* and *I2BSP* to convert between bit strings and integers are defined as follows.

The function *BS2IP*(x) maps a bit string x to an integer value x' , as follows. If $x = \langle x_{l-1}, \dots, x_0 \rangle$ where $x_0, \dots, x_{l-1}$ are bits, then the value x' is defined as

$$x' = \sum_{\substack{0 \leq i < l \\ x_i = '1'}} 2^i.$$

The function *I2BSP*(m, l) takes as input two non-negative integers m and l , and outputs the unique bit string x of length l such that *BS2IP*(x) = m , if such an x exists. Otherwise, the function **fails**.

The *length in bits of a non-negative integer n* is the number of bits in its binary representation, i.e., $\lceil \log_2(n+1) \rceil$.

As a notational convenience, *Oct*(m) is defined as *Oct*(m) = *I2BSP*($m, 8$).

NOTE Note that *I2BSP*(m, l) **fails** if and only if the length of m in bits is greater than l .

5.2.5 Octet string/integer conversion

Primitives *OS2IP* and *I2OSP* to convert between octet strings and integers are defined as follows.

The function *OS2IP*(x) takes as input an octet string, and outputs the integer *BS2IP*(*OS2BSP*(x)).

The function *I2OSP*(m, l) takes as input two non-negative integers m and l , and outputs the unique octet string x of length l such that *OS2IP*(x) = m , if such an x exists. Otherwise, the function **fails**.

The *length in octets of a non-negative integer n* is the number of digits in its representation base 256, i.e., $\lceil \log_{256}(n+1) \rceil$; this quantity is denoted $\mathcal{L}(n)$.

NOTE Note that *I2OSP*(m, l) **fails** if and only if the length of m in octets is greater than l .

5.3 Finite fields

This clause describes a very general framework for describing specific finite fields. A finite field specified in this way is called an *explicitly given finite field*, and it is determined by *explicit data*.

For a finite field F of cardinality $q = p^e$, where p is prime and $e \geq 1$, explicit data for F consists of p and e , along with a “multiplication table,” which is a matrix $T = (T_{ij})_{1 \leq i, j \leq e}$, where each T_{ij} is an e -tuple over $[0..p)$.

The set of elements of F is the set of all e -tuples over $[0..p)$. The entries of T are themselves viewed as elements of F .

Addition in F is defined element-wise: if

$$a = (a_1, \dots, a_e) \in F \quad \text{and} \quad b = (b_1, \dots, b_e) \in F,$$

then $a + b = c$, where

$$c = (c_1, \dots, c_e) \quad \text{and} \quad c_i = (a_i + b_i) \bmod p \quad (1 \leq i \leq e).$$

A scalar multiplication operation for F is also defined element-wise: if

$$a = (a_1, \dots, a_e) \in F \quad \text{and} \quad d \in [0..p),$$

then $d \cdot a = c$, where

$$c = (c_1, \dots, c_e) \quad \text{and} \quad c_i = (d \cdot a_i) \bmod p \quad (1 \leq i \leq e).$$

Multiplication in F is defined via the multiplication table T , as follows: if

$$a = (a_1, \dots, a_e) \in F \quad \text{and} \quad b = (b_1, \dots, b_e) \in F,$$

$$a \cdot b = \sum_{i=1}^e \sum_{j=1}^e (a_i b_j \bmod p) T_{ij},$$

where the products $(a_i b_j \bmod p) T_{ij}$ are defined using the above rule for scalar multiplication, and where these products are summed using the above rule for addition in F . It is assumed that the multiplication table defines an algebraic structure that satisfies the usual axioms of a field; in particular, there exist additive and multiplicative identities, every element has an additive inverse, and every element besides the additive identity has a multiplicative inverse.

Observe that the additive identity of F , denoted 0_F , is the all-zero e -tuple, and that the multiplicative identity of F , denoted 1_F , is a non-zero e -tuple whose precise format depends on T .

NOTE 1 The field F is a vector space of dimension e over the prime field F' of cardinality p , where scalar multiplication is defined as above. The prime p is called the *characteristic* of F . For $1 \leq i \leq e$, let θ_i denote the e -tuple over F' whose i th component is 1, and all of whose other components are 0. The elements $\theta_1, \dots, \theta_e$ form an ordered basis of F as a vector space over F' . Note that for $1 \leq i, j \leq e$, we have $\theta_i \cdot \theta_j = T_{ij}$.

NOTE 2 For $e > 1$, two types of *standard bases* are defined that are commonly used in implementations of finite field arithmetic:

- $\theta_1, \dots, \theta_e$ is called a *polynomial basis* for F over F' if for some $\theta \in F$, $\theta_i = \theta^{e-i}$ for $1 \leq i \leq e$. Note that in this case, $1_F = \theta_e$.
- $\theta_1, \dots, \theta_e$ is called a *normal basis* for F over F' if for some $\theta \in F$, $\theta_i = \theta^{p^{i-1}}$ for $1 \leq i \leq e$. Note that in this case, $1_F = c \sum_{i=1}^e \theta_i$ for some $c \in [1..p)$; if $p = 2$, then the only possible choice for c is 1; moreover, one can always choose a normal basis for which $c = 1$.

NOTE 3 The definition given here of an explicitly given finite field comes from [23].

5.3.1 Octet string and integer/finite field conversion

Primitives $OS2FEP_F$ and $FE2OSP_F$ to convert between octet strings and elements of an explicitly given finite field F , as well as the primitive $FE2IP_F$ to convert elements of F to integer values, are defined as follows.

The function $FE2IP_F$ maps an element $a \in F$ to an integer value a' , as follows. If the cardinality of F is $q = p^e$, where p is prime and $e \geq 1$, then an element a of F is an e -tuple $(a_1, \dots, a_e)$, where $a_i \in [0 \dots p)$ for $1 \leq i \leq e$, and the value a' is defined as

$$a' = \sum_{i=1}^e a_i p^{i-1}.$$

The function $FE2OSP_F(a)$ takes as input an element a of the field F and outputs the octet string $I2OSP(a', l)$, where $a' = FE2IP_F(a)$, and l is the length in octets of $|F| - 1$, i.e., $l = \lceil \log_{256} |F| \rceil$. Thus, the output of $FE2OSP_F(a)$ is always an octet string of length exactly $\lceil \log_{256} |F| \rceil$.

The function $OS2FEP_F(x)$ takes as input an octet string x , and outputs the (unique) field element $a \in F$ such that $FE2OSP_F(a) = x$, if any such a exists, and otherwise **fails**. Note that $OS2FEP_F(x)$ **fails** if and only if either x does not have length exactly $\lceil \log_{256} |F| \rceil$, or $OS2IP(x) \geq |F|$.

5.4 Elliptic curves

An elliptic curve E over an explicitly given finite field F is a set of points $P = (x, y)$, where x and y are elements of F that satisfy a certain equation, together with the “point at infinity,” denoted by $\mathcal{O}$. For the purposes of this part of ISO/IEC 18033, the curve E is specified by two field elements $a, b \in F$, called the *coefficients* of E .

Let p be the characteristic of F .

If $p > 3$, then a and b shall satisfy $4a^3 + 27b^2 \neq 0_F$, and every point $P = (x, y)$ on E (other than $\mathcal{O}$) shall satisfy the equation

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

If $p = 2$, then b shall satisfy $b \neq 0_F$, and every point $P = (x, y)$ on E (other than $\mathcal{O}$) shall satisfy the equation

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

If $p = 3$, then a and b shall satisfy $a \neq 0_F$ and $b \neq 0_F$, and every point $P = (x, y)$ on E (other than $\mathcal{O}$) shall satisfy the equation

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

The points on an elliptic curve form a finite abelian group, where $\mathcal{O}$ is the identity element. There exist efficient algorithms to perform the group operation of an elliptic curve, but the implementation of such algorithms is out of the scope of this part of ISO/IEC 18033.

NOTE See, for example, ISO/IEC 15946-1, as well as [9], for more information on how to efficiently implement elliptic curve group operations.

5.4.1 Compressed elliptic curve points

Let E be an elliptic curve over an explicitly given finite field F , where F has characteristic p .

A point $P \neq \mathcal{O}$ can be represented in either *compressed*, *uncompressed*, or *hybrid* form.

If $P = (x, y)$, then (x, y) is the uncompressed form of P .

Let $P = (x, y)$ be a point on the curve E , as above. The *compressed form* of P is the pair $(x, \tilde{y})$, where $\tilde{y} \in \{0, 1\}$ is determined as follows.

- If $p \neq 2$ and $y = 0_F$, then $\tilde{y} = 0$.
- If $p \neq 2$ and $y \neq 0_F$, then $\tilde{y} = ((y'/p^f) \bmod p) \bmod 2$, where $y' = FE2IP_F(y)$, and where f is the largest non-negative integer such that $p^f \mid y'$.
- If $p = 2$ and $x = 0_F$, then $\tilde{y} = 0$.
- If $p = 2$ and $x \neq 0_F$, then $\tilde{y} = \lfloor z'/2^f \rfloor \bmod 2$, where $z = y/x$, where $z' = FE2IP_F(z)$, and where f is the largest non-negative integer such that 2^f divides $FE2IP_F(1_F)$.

The *hybrid form* of $P = (x, y)$ is the triple $(x, \tilde{y}, y)$, where $\tilde{y}$ is as in the previous paragraph.

5.4.2 Point decompression algorithms

There exist efficient procedures for *point decompression*, i.e., computing y from $(x, \tilde{y})$. These are briefly described here.

- Assume $p \neq 2$, and let $(x, \tilde{y})$ be the compressed form of (x, y) . The point (x, y) satisfies an equation $y^2 = f(x)$ for a polynomial $f(x)$ over F in x . If $f(x) = 0_F$, then there is only one possible choice for y , namely, $y = 0_F$. Otherwise, if $f(x) \neq 0$, then there are two possible choices of y , which differ only in sign, and the correct choice is determined by $\tilde{y}$. There are well-known algorithms for computing square roots in finite fields, and so the two choices of y are easily computed.
- Assume $p = 2$, and let $(x, \tilde{y})$ be the compressed form of (x, y) . The point (x, y) satisfies an equation $y^2 + xy = x^3 + ax^2 + b$. If $x = 0_F$, then we have $y^2 = b$, from which y is uniquely determined and easily computed. Otherwise, if $x \neq 0_F$, then setting $z = y/x$, we have $z^2 + z = g(x)$, where $g(x) = (x + a + bx^{-2})$. The value of y is uniquely determined by, and easily computed from, the values z and x , and so it suffices to compute z . To compute z , observe that for a fixed x , if z is one solution to the equation $z^2 + z = g(x)$, then there is exactly one other solution, namely $z + 1_F$. It is easy to compute these two candidate values of z , and the correct choice of z is easily seen to be determined by $\tilde{y}$.

5.4.3 Octet string/elliptic curve conversion

Primitives $EC2OSP_E$ and $OS2ECP_E$ for converting between points on an elliptic curve E and octet strings are defined as follows.

Let E be an elliptic curve over an explicitly given finite field F .

The function $EC2OSP_E(P, fmt)$ takes as input a point P on E and a format specifier fmt , which is one of the symbolic values **compressed**, **uncompressed**, or **hybrid**. The output is an octet string EP , computed as follows.

- If $P = \mathcal{O}$, then $EP = \langle Oct(0) \rangle$.
- If $P = (x, y) \neq \mathcal{O}$, with compressed form $(x, \tilde{y})$, then

$$EP = \langle H \rangle \| X \| Y,$$
 where
 - H is a single octet of the form $Oct(4U + C \cdot (2 + \tilde{y}))$, where
 - $U = 1$ if fmt is either **uncompressed** or **hybrid**, and otherwise, $U = 0$;
 - $C = 1$ if fmt is either **compressed** or **hybrid**, and otherwise, $C = 0$;
 - X is the octet string $FE2OSP_F(x)$;
 - Y is the octet string $FE2OSP_F(y)$ if fmt is either **uncompressed** or **hybrid**, and otherwise Y is the null octet string.

NOTE If the format specifier fmt is **uncompressed**, then the value $\tilde{y}$ need not be computed.

The function $OS2ECP_E(EP)$ takes as input an octet string EP . If there exists a point P on the curve E and a format specifier fmt such that $EC2OSP_E(P, fmt) = EP$, then the function outputs P (in uncompressed form), and otherwise, the function **fails**. Note that the point P , if it exists, is uniquely defined, and so the function $OS2ECP_E(EP)$ is well defined.

6 Cryptographic transformations

This clause describes several cryptographic transformations that will be referred to in subsequent clauses. The types of transformations are *cryptographic hash functions*, *key derivation functions*, *message authentication codes*, *block ciphers*, and *symmetric ciphers*. For each type of transformation, the abstract input/output characteristics are given, and then specific implementations of these transformations that are allowed for use in this part of ISO/IEC 18033 are specified.

6.1 Cryptographic hash functions

A cryptographic hash function is essentially a function that maps an octet string of variable length to an octet string of fixed length. More precisely, a cryptographic hash function $Hash$ specifies

- a positive integer $Hash.len$ that denotes the length of the hash function output,
- a positive integer $Hash.MaxInputLen$ that denotes the maximum length hash input,
- and a function $Hash.eval$ that denotes the hash function itself, which maps octet strings of length at most $Hash.MaxInputLen$ to octet strings of length $Hash.len$.

The invocation of $Hash.eval$ **fails** if and only if the input length exceeds $Hash.MaxInputLen$.

6.1.1 Allowable cryptographic hash functions

For the purposes of this part of ISO/IEC 18033, the allowable cryptographic hash functions are those described in ISO/IEC 10118-2 and ISO/IEC 10118-3, with the following provisos:

- The hash functions described in ISO/IEC 10118 map bit strings to bit strings, whereas in this part of ISO/IEC 18033, they map octet strings to octet strings. Therefore, a hash function in ISO/IEC 10118-2 or ISO/IEC 10118-3 is allowed in this part of ISO/IEC 18033 only if the length in bits of the output is a multiple of 8, in which case the mapping between octet strings and bit strings is affected by the functions *OS2BSP* and *BS2OSP*.
- Whereas the hash functions in ISO/IEC 10118 are not defined for inputs exceeding a given length, a hash function in this part of ISO/IEC 18033 is defined to **fail** for such inputs.

6.2 Key derivation functions

A *key derivation function* is a function $KDF(x, l)$ that takes as input an octet string x and an integer $l \geq 0$, and outputs an octet string of length l . The string x is of arbitrary length, although an implementation may define a (very large) maximum length for x and maximum size for l , and **fail** if these bounds are exceeded.

NOTE In some other documents and standards, the term “mask generation function” is used instead of “key derivation function.”

6.2.1 Allowable key derivation functions

The key derivation functions that are allowed in this part of ISO/IEC 18033 are *KDF1*, described below in Clause 6.2.2, and *KDF2*, described below in Clause 6.2.3.

6.2.2 *KDF1*

6.2.2.1 System parameters

KDF1 is a family of key derivation functions, parameterized by the following system parameters:

- *Hash*: a cryptographic hash function, as described in Clause 6.1.

6.2.2.2 Specification

For an octet string x and a non-negative integer l , $KDF1(x, l)$ is defined to be the first l octets of

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

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

NOTE This function will **fail** if and only if $k > 2^{32}$ or if $|x| + 4 > Hash.MaxInputLen$.

6.2.3 *KDF2*

6.2.3.1 System parameters

KDF2 is a family of key derivation functions, parameterized by the following system parameters:

- *Hash*: a cryptographic hash function, as described in Clause 6.1.

6.2.3.2 Specification

For an octet string x and a non-negative integer l , $KDF2(x, l)$ is defined to be the first l octets of

$$Hash.eval(x \parallel I2OSP(1, 4)) \parallel \cdots \parallel Hash.eval(x \parallel I2OSP(k, 4)),$$

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

NOTE 1 This function will **fail** if and only if $k \geq 2^{32}$ or if $|x| + 4 > Hash.MaxInputLen$.

NOTE 2 $KDF2$ is the same as $KDF1$, except that the counter runs from 1 to k , rather than from 0 to $k - 1$.

6.3 MAC algorithms

A MAC algorithm MA is a scheme that defines two positive integers $MA.KeyLen$ and $MA.MACLen$, along with a function $MA.eval(k', T)$ that takes a secret key k' , which is an octet string of length $MA.KeyLen$, along with an arbitrary octet string T as input, and computes as output an octet string MAC of length $MA.MACLen$.

An implementation may impose a maximum value for the length of T , and $MA.eval(k', T)$ will **fail** if this bound is exceeded.

NOTE See Annex B.1 for a discussion on the desired security properties of MAC algorithms.

6.3.1 Allowable MAC algorithms

For the purposes of this part of ISO/IEC 18033, the allowable MAC algorithms are those described in ISO/IEC 9797-1 and ISO/IEC 9797-2, with the following provisos:

- For MAC the algorithms described in ISO/IEC 9797-1 and ISO/IEC 9797-2, the inputs are bit strings, and the secret key and outputs are fixed-length bit strings. Therefore, an algorithm in ISO/IEC 9797-1 or ISO/IEC 9797-2 is allowed in this part of ISO/IEC 18033 only if the lengths in bits of the MAC and of the secret key are multiples of 8, in which case the mapping between octet strings and bit strings is affected by the functions $OS2BSP$ and $BS2OSP$.
- Whereas the algorithms in ISO/IEC 9797-1 and ISO/IEC 9797-2 are not defined for inputs exceeding a given length, a MAC algorithm in this part of ISO/IEC 18033 is defined to **fail** for such inputs.

6.4 Block ciphers

A block cipher BC specifies the following:

- a positive integer $BC.KeyLen$, which is the length in octets of the secret key,
- a positive integer $BC.BlockLen$, which is the length in octets of a block of plaintext or ciphertext,
- a function $BC.Encrypt(k, b)$, which takes as input a secret key k , which is an octet string of length $BC.KeyLen$, and a plaintext block b , which is an octet string of length $BC.BlockLen$, and outputs a ciphertext block b' , which is an octet string of length $BC.BlockLen$, and

- a function $BC.Decrypt(k, b')$, which takes as input a secret key k , which is an octet string of length $BC.KeyLen$, and a ciphertext block b' , which is an octet string of length $BC.BlockLen$, and outputs a plaintext block b , which is an octet string of length $BC.BlockLen$.

For any fixed secret key k , the function $b \mapsto BC.Encrypt(k, b)$ acts as a permutation on the set of octet strings of length $BC.BlockLen$, and the function $b' \mapsto BC.Decrypt(k, b')$ acts as the inverse permutation.

NOTE See Annex B.2 for a discussion of the desired security properties of block ciphers.

6.4.1 Allowable block ciphers

For the purposes of this part of ISO/IEC 18033, the allowable block ciphers are those described in ISO/IEC 18033-3, with the following proviso:

- In ISO/IEC 18033-3, plaintext/ciphertext blocks and secret keys are fixed-length bit strings, whereas in this part of ISO/IEC 18033, they are fixed-length octet strings. Therefore, a block cipher in ISO/IEC 18033-3 is allowed in this part of ISO/IEC 18033 only if the lengths in bits of plaintext/ciphertext blocks and of the secret key are multiples of 8, in which case the mapping between octet strings and bit strings is affected by the functions $OS2BSP$ and $BS2OSP$.

6.5 Symmetric ciphers

A symmetric cipher SC specifies a key length $SC.KeyLen$, along with encryption and decryption algorithms:

- The encryption algorithm $SC.Encrypt(k, M)$ takes as input a secret key k , which is an octet string of length $SC.KeyLen$, and a plaintext M , which is an octet string of arbitrary length. It outputs a ciphertext c , which is an octet string.

The encryption algorithm may **fail** if the length of M exceeds some large, implementation-defined limit.

- The decryption algorithm $SC.Decrypt(k, c)$ takes as input a secret key k , which is an octet string of length $SC.KeyLen$, and a ciphertext c , which is an octet string of arbitrary length. It outputs a plaintext M , which is an octet string.

The decryption algorithm may **fail** under some circumstances.

The encryption and decryption algorithms are deterministic. Also, for all secret keys k and all plaintexts M , if M does not exceed the length bound of the encryption algorithm, and if $c = SC.Encrypt(k, M)$, then $SC.Decrypt(k, c)$ does not **fail** and $SC.Decrypt(k, c) = M$.

NOTE See Annex B.3 for a discussion on the desired security properties for a symmetric cipher.

6.5.1 Allowable symmetric ciphers

The symmetric ciphers that are allowed in this part of ISO/IEC 18033 are

- $SC1$, described below in Clause 6.5.2, and
- $SC2$, described below in Clause 6.5.3.

6.5.2 SC1

This symmetric cipher is the cipher obtained by using a block cipher in a particular cipher block chaining (CBC) mode (see ISO/IEC 10116), together with a particular padding scheme to pad cleartexts so that their length is a multiple of the block size of the underlying block cipher.

6.5.2.1 System parameters

SC1 is a family of symmetric ciphers, parameterized by the following system parameters:

— *BC*: a block cipher, as described in Clause 6.4.

Strictly speaking, one must make the restriction that $BC.BlockLen < 256$; however, in practice this restriction is always met.

6.5.2.2 Specification

$SC1.KeyLen = BC.KeyLen$.

The function $SC1.Encrypt(k, M)$ works as follows.

- a) Set $padLen = BC.BlockLen - (|M| \bmod BC.BlockLen)$.
- b) Let $P_1 = Oct(padLen)$.
- c) Let P_2 be the octet string formed by repeating the octet P_1 a total of $padLen$ times (so $|P_2| = padLen$).
- d) Let $M' = M \parallel P_2$.
- e) Parse M' as $M'_1 \parallel \dots \parallel M'_l$, where for $1 \leq i \leq l$, M'_i is an octet string of length $BC.BlockLen$.
- f) Let c_0 be the octet string consisting of $BC.BlockLen$ copies of the octet $Oct(0)$, and for $1 \leq i \leq l$, let $c_i = BC.Encrypt(k, M'_i \oplus c_{i-1})$.
- g) Let $c = c_1 \parallel \dots \parallel c_l$.
- h) Output c .

The function $SC1.Decrypt(k, c)$ works as follows.

- a) If $|c|$ is not a non-zero multiple of $BC.BlockLen$, then **fail**.
- b) Parse c as $c = c_1 \parallel \dots \parallel c_l$, where for $1 \leq i \leq l$, c_i is an octet string of length $BC.BlockLen$. Also, let c_0 be the octet string consisting of $BC.BlockLen$ copies of the octet $Oct(0)$.
- c) For $1 \leq i \leq l$, let $M'_i = BC.Decrypt(k, c_i) \oplus c_{i-1}$.
- d) Let P_1 be the last octet of M'_l , and let $padLen = BS2IP(P_1)$.

- e) If $padLen \notin [1..BC.BlockLen]$, then **fail**.
- f) Check that the last $padLen$ octets of M'_l are equal to P_1 ; if not, then **fail**.
- g) Let M''_l be the octet string consisting of the first $BC.BlockLen - padLen$ octets of M'_l .
- h) Set $M = M'_1 \parallel \dots \parallel M'_{l-1} \parallel M''_l$.
- i) Output M .

6.5.3 SC2

6.5.3.1 System parameters

SC2 is a family of symmetric ciphers, parameterized by the following system parameters:

- *KDF*: a key derivation function, as described in Clause 6.2;
- *KeyLen*: a positive integer.

6.5.3.2 Specification

The value of *SC2.KeyLen* is equal to the value of the system parameter *KeyLen*.

The function *SC2.Encrypt*(k, M) works as follows.

- a) Set $mask = KDF(k, |M|)$.
- b) Set $c = mask \oplus M$.
- c) Output c .

The function *SC2.Decrypt*(k, c) works as follows.

- a) Set $mask = KDF(k, |c|)$.
- b) Set $M = mask \oplus c$.
- c) Output M .

7 Asymmetric ciphers

An asymmetric cipher *AC* consists of three algorithms:

- A key generation algorithm *AC.KeyGen*($\cdot$), that outputs a public-key/private-key pair (PK, pk). The structure of PK and pk depends on the particular cipher.
- An encryption algorithm *AC.Encrypt*(PK, L, M, opt) that takes as input a public key PK , a label L , a plaintext M , and an encryption option opt , and outputs a ciphertext C . Note that L , M , and C are octet strings. See Clause 7.2 below for more on *labels*. See Clause 7.4 below for more on *encryption options*.

The encryption algorithm may **fail** if the lengths L or M exceed some implementation-defined limits.

- A decryption algorithm $AC.Decrypt(pk, L, C)$ that takes as input a private key pk , a label L , and a ciphertext C , and outputs a plaintext M .

The decryption algorithm may **fail** under some circumstances.

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

NOTE 1 The intent is that all of the asymmetric ciphers described in this part of ISO/IEC 18033 provide reasonable security against adaptive chosen ciphertext attack (as defined in [30], and which is equivalent to a notion of “non-malleability” defined in [17]). This notion of security is generally regarded by the cryptographic research community as the appropriate form of security that a general-purpose asymmetric cipher should provide. The formal definition of this notion of security is presented in Annex B.4, appropriately adapted to take into account variable length plaintexts and the role of *labels*; also, a slightly weaker notion of security, called “benign malleability,” is defined. This notion of “benign malleability” is also adequate for most, if not all, applications of asymmetric ciphers, and some of the asymmetric ciphers described in this part of ISO/IEC 18033 only achieve this level of security.

NOTE 2 A basic requirement of any asymmetric cipher is *correctness*: for any public-key/private-key pair (PK, pk) , for any label/plaintext pair (L, M) , such that the lengths of L and M do not exceed the implementation-defined limits, any encryption of M with label L under PK decrypts with label L under pk to the original plaintext M . This requirement may be relaxed, so that it holds only for all but a negligible fraction of public-key/private-key pairs.

NOTE 3 As an example of an asymmetric cipher AC for which the above correctness requirement may not always hold, consider any RSA-based cipher where the modulus $n = pq$, where p and q should be prime. The key generation algorithm may use a probabilistic algorithm for testing if p and q are prime, and this algorithm may produce incorrect results with a negligible probability; if this happens, the decryption algorithm may not be the inverse of the encryption algorithm.

7.1 Plaintext length

It is important to note that plaintexts may be of arbitrary and variable length, although an implementation may impose a (typically, very large) upper bound on this length.

However, two degenerate types of asymmetric ciphers are defined as follows:

- A *fixed-plaintext-length* asymmetric cipher AC only encrypts plaintexts whose length (in octets) is equal to a fixed value $AC.MsgLen$.
- A *bounded-plaintext-length* asymmetric cipher AC only encrypts plaintexts whose length (in octets) is less than or equal to a fixed value $AC.MaxMsgLen(PK)$. Here, the maximum plaintext length may depend on the public key PK of the cipher.

NOTE Except for fixed-plaintext-length and bounded-plaintext-length asymmetric ciphers, the encryption of a plaintext will in general not hide the length of the plaintext. Therefore, it is up to the application using the asymmetric cipher to ensure, perhaps by an appropriate padding scheme, that no sensitive information is implicitly encoded in the length of a plaintext.

7.2 The use of labels

A *label* is an octet string whose value is used by the encryption and decryption algorithms. It may contain public data that is implicit from context and need not be encrypted, but that should nevertheless be bound to the ciphertext.

A label is an octet string that is meaningful to the application using the asymmetric cipher, and that is independent of the implementation of the asymmetric cipher.

Labels may be of arbitrary and variable length, although a particular cipher may choose to impose a (very large) upper bound on this length.

A degenerate type of asymmetric cipher is defined as follows:

- A *fixed-label-length* asymmetric cipher is one in which the encryption and decryption algorithms only accept labels whose length (in octets) is equal to a fixed value $AC.LabelLen$.

NOTE 1 The traditional notion of security against adaptive chosen ciphertext attack has been extended in Annex B.4, so that intuitively, for a secure asymmetric cipher, the encryption algorithm should bind the label to the ciphertext in an appropriate “non-malleable” fashion.

NOTE 2 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 plaintext. 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.

7.3 Ciphertext format

The asymmetric ciphers proposed in this part of ISO/IEC 18033 describe precisely how a ciphertext is to be formatted as an octet string. However, an implementation is free to store and/or transmit ciphertexts in alternative formats, if this is convenient. Moreover, the process of encrypting a plaintext and converting the resulting ciphertext into an alternative format may be collapsed into a single, functionally equivalent process; likewise, the process of converting from an alternative format and decrypting the ciphertext may be collapsed into a single, functionally equivalent process. Thus, in a given system, ciphertexts need never appear in the format prescribed here.

NOTE Besides promoting inter-operability, prescribing the format of a ciphertext is necessary in order to make meaningful claims and to reason about the security of an asymmetric cipher against adaptive chosen ciphertext attacks.

7.4 Encryption options

Some asymmetric ciphers allow certain types of scheme-specific options to be passed to the encryption algorithm, which is why an extra encryption option argument *opt* is allowed in the abstract interface for an asymmetric cipher.

Some asymmetric ciphers presented here may naturally be viewed as not having any encryption options, in which case, the cipher is said to take no encryption option.

A system may provide a “default” value of *opt*; however, such provisions are outside the scope of this part of ISO/IEC 18033.

NOTE Among the specific asymmetric ciphers described in this part of ISO/IEC 18033, only the elliptic-curve-based ciphers use an encryption option, which is used to indicate the desired format for encoding points on elliptic curves.

7.5 Method of operation of an asymmetric cipher

Typically, the key generation algorithm is run by some party, known as the *owner* of the key pair, or by some trusted party on the owner’s behalf. The public key shall be made available to all parties who wish to send encrypted messages to the owner, while the private key shall not be divulged to any party other than the owner. Mechanisms and protocols for making a public key available to other parties are out of the scope of this part of ISO/IEC 18033. See ISO/IEC 11770 for guidance on this issue.

Each of the asymmetric ciphers presented in this part of ISO/IEC 18033 are actually members of *families* of asymmetric ciphers, where a particular cipher is selected from the family by choosing particular values for the *system parameters* defining the family of ciphers.

For a cipher selected from a family of ciphers, prior to key generation, specific values of the system parameters for the family shall be chosen. Depending on the conventions used for encoding public keys, some of the choices of the system parameters may be embedded in the encoding of the public key as well. These system parameters shall remain fixed throughout the lifetime of the public key.

NOTE For example, if an asymmetric cipher may be parameterized in terms of a cryptographic hash function, the choice of hash function should be fixed once and for all at some point prior to the generation of a public-key/private-key pair, and the encryption and decryption algorithms should use the chosen hash function throughout the lifetime of the public key. Failure to abide by this rule not only makes an implementation non-conforming, but also invalidates the security analysis for the cipher, and may in some cases expose the implementation to severe security risks.

7.6 Allowable asymmetric ciphers

Users who wish to employ an asymmetric cipher from this part of ISO/IEC 18033 shall select one of the following:

- a generic hybrid cipher chosen from the family *HC* of hybrid ciphers described in Clause 8.3;
- a bounded-plaintext-length asymmetric cipher from the family *RSAES* of ciphers described in Clause 11.4;
- a bounded-plaintext-length asymmetric cipher from the family *HIME(R)* of ciphers described in Clause 12.3.

NOTE As each of *HC*, *RSAES*, and *HIME(R)* are families of ciphers, parameterized by various system parameters, a user will have to choose specific values of these system parameters from the set of allowable system parameters specified in the corresponding clause in which each family is described.

8 Generic hybrid ciphers

In designing an efficient asymmetric cipher, a useful approach is to design a *hybrid cipher*, where one uses asymmetric cryptographic techniques to encrypt a secret key that can then be used to

encrypt the actual message using symmetric cryptographic techniques. This clause describes a specific type of hybrid cipher, called a *generic hybrid cipher*. A generic hybrid cipher is built from two lower-level “building blocks”: a *key encapsulation mechanism* and a *data encapsulation mechanism*. Clause 8.3 specifies in detail the family *HC* of generic hybrid ciphers.

8.1 Key encapsulation mechanisms

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

- A key generation algorithm *KEM.KeyGen()*, that outputs a public-key/private-key pair (PK, pk) . The structure of PK and pk depends on the particular scheme.
- An encryption algorithm *KEM.Encrypt*(PK, opt) that takes as input a public key PK , along with an encryption option opt , and outputs a secret-key/ciphertext pair (K, C_0) . Both K and C_0 are octet strings. The role of opt is analogous to its role in asymmetric ciphers (see Clause 7.4).
- A decryption algorithm *KEM.Decrypt*(pk, C_0) that takes as input a private key pk and a ciphertext C_0 , and outputs a secret key K . Both K and C_0 are octet strings.

The decryption algorithm may **fail** under some circumstances.

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

NOTE Any key encapsulation mechanism should satisfy a correctness property analogous to the correctness property of an asymmetric cipher: for any public-key/private-key pair (PK, pk) , for any output (K, C_0) of the encryption algorithm on input (PK, opt) , the ciphertext C_0 should decrypt under pk to K . This requirement may be relaxed, so that it holds only for all but a negligible fraction of public-key/private-key pairs.

8.1.1 Prefix-freeness property

Additionally, a key encapsulation mechanism must satisfy the following property. The set of all possible ciphertext outputs of the encryption algorithm should be a subset of a *candidate* set of octet strings (that may depend on the public key), such that the candidate set is prefix free and elements of the candidate set are easy to recognize (given either the public key or the private key).

8.1.2 Allowable key encapsulation mechanisms

The key encapsulation mechanisms that are allowed in this part of ISO/IEC 18033 are

- *ECIES-KEM* (described in Clause 10.2),
- *PSEC-KEM* (described in Clause 10.3),
- *ACE-KEM* (described in Clause 10.4), and
- *RSA-KEM* (described in Clause 11.5).

NOTE 1 As a matter of convention, the corresponding generic hybrid ciphers built from these key encapsulation mechanisms via the generic hybrid construction in Clause 8.3 shall be called (respectively) *ECIES-HC*, *PSEC-HC*, *ACE-HC*, and *RSA-HC*.

NOTE 2 Roughly speaking, a key encapsulation mechanism works just like an asymmetric cipher, except that the encryption algorithm takes no input other than the recipient's public key: instead of taking a message as input and producing a ciphertext, the encryption algorithm generates a secret-key/ciphertext pair (K, C_0) , where K is an octet string of some specified length, and C_0 is an encryption of K , that is, the decryption algorithm applied to C_0 yields K .

NOTE 3 One can always use a (possibly fixed-plaintext-length or bounded-plaintext-length) asymmetric cipher for this purpose, generating a random octet string K , and then encrypting it under the recipient's public key (and any encryption options) to obtain C_0 . However, one can construct a key encapsulation mechanism in other, more efficient, ways as well.

NOTE 4 For the purposes of building a generic hybrid cipher that is secure against adaptive chosen ciphertext attack, there is a corresponding notion of security for a key encapsulation mechanism. This is discussed in detail in Annex B.5.

8.2 Data encapsulation mechanisms

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 secret key K , a label L , and a plaintext M . It outputs a ciphertext C_1 . Here, K , L , M , and C_1 are octet strings, and L and M may have arbitrary length, and K is of length $DEM.KeyLen$.

The encryption algorithm may **fail** if the lengths L or M exceed some (very large) implementation-defined limits.

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

The decryption algorithm may **fail** under some circumstances.

NOTE The encryption and decryption algorithms should be deterministic, and should satisfy the following correctness requirement: for all secret keys K , all labels L , and all plaintexts M , such that the lengths of L and M do not exceed the implementation-defined limits,

$$DEM.Decrypt(K, L, DEM.Encrypt(K, L, M)) = M.$$

8.2.1 Degenerate types of data encapsulation mechanisms

Two different, degenerate types of data encapsulation mechanisms are defined as follows:

- A *fixed-label-length* data encapsulation mechanism is one for which the encryption and decryption algorithms only accept labels whose lengths are equal to a fixed value $DEM.LabelLen$.
- A *fixed-plaintext-length* data encapsulation mechanism is one for which the encryption algorithm only accepts plaintexts whose lengths are equal to a fixed value $DEM.MsgLen$.

8.2.2 Allowable data encapsulation mechanisms

The data encapsulation mechanisms that are allowed in this part of ISO/IEC 18033 are described in Clause 9.

NOTE 1 Roughly speaking, a data encapsulation mechanism provides a “digital envelope” that protects both the confidentiality and integrity of data using symmetric cryptographic techniques; it may also bind the data to a public label.

NOTE 2 For the purposes of building a generic hybrid cipher that is secure against adaptive chosen ciphertext attack, there is a corresponding notion of security for a data encapsulation mechanism. This is discussed in detail in Annex B.6.

8.3 *HC*

8.3.1 System parameters

HC is a family of asymmetric ciphers parameterized by the following system parameters:

- *KEM*: a key encapsulation mechanism, as described in Clause 8.1;
- *DEM*: a data encapsulation mechanism, as described in Clause 8.2.

Any combination of *KEM* and *DEM* may be used, provided $KEM.KeyLen = DEM.KeyLen$.

NOTE 1 If *DEM* is a fixed-label-length data encapsulation mechanism, with labels restricted to length $DEM.LabelLen$, then *HC* is a fixed-label-length asymmetric cipher with $HC.LabelLen = DEM.LabelLen$.

NOTE 2 If *DEM* is a fixed-plaintext-length data encapsulation mechanism, with plaintexts restricted to length $DEM.MsgLen$, then *HC* is a fixed-plaintext-length asymmetric cipher with $HC.MsgLen = DEM.MsgLen$.

NOTE 3 For all the allowable choices of *KEM*, the value of $KEM.KeyLen$ is a system parameter that may be chosen so as to equal $DEM.KeyLen$. Thus, all possible combinations of allowable *KEM* and *DEM* may be realized by appropriate choices of system parameters.

8.3.2 Key generation

The key generation algorithm, public key, and private key for *HC* are the same as that of *KEM*. The encryption options of *HC* are the same as that of *KEM*.

Let (PK, pk) denote a public-key/private-key pair.

8.3.3 Encryption

The encryption algorithm *HC.Encrypt* takes as input a public key *PK*, a label *L*, a plaintext *M*, and an encryption option *opt*. It runs as follows.

- a) Compute $(K, C_0) = KEM.Encrypt(PK, opt)$.
- b) Compute $C_1 = DEM.Encrypt(K, L, M)$.
- c) Set $C = C_0 \parallel C_1$.
- d) Output *C*.

8.3.4 Decryption

The decryption algorithm *HC.Decrypt* takes as input a private key pk , a label L , and a ciphertext C . It runs as follows.

- a) Using the prefix-freeness property of the ciphertexts associated with *KEM* (see Clause 8.1.1), parse C as $C = C_0 \| C_1$, where C_0 and C_1 are octet strings such that C_0 is an element of the candidate set of possible ciphertexts associated with *KEM*. This step **fails** if C cannot be so parsed.
- b) Compute $K = KEM.Decrypt(pk, C_0)$.
- c) Compute $M = DEM.Decrypt(K, L, C_1)$
- d) Output M .

NOTE The security of *HC* is discussed in Annex B.7. It is only remarked here that so long as *KEM* and *DEM* satisfy the appropriate security properties, then *HC* will be secure against adaptive chosen ciphertext attack.

9 Constructions of data encapsulation mechanisms

This clause specifies the data encapsulation mechanisms that are allowed in this part of ISO/IEC 18033. These mechanisms are

- *DEM1*, described below in Clause 9.1,
- *DEM2*, described below in Clause 9.2, and
- *DEM3*, described below in Clause 9.3.

9.1 *DEM1*

9.1.1 System parameters

DEM1 is a family of data encapsulation mechanisms, parameterized by the following system parameters:

- *SC*: a symmetric cipher, as described in Clause 6.5;
- *MA*: a MAC algorithm, as described in Clause 6.3.

The value of *DEM1.KeyLen* is defined as $DEM1.KeyLen = SC.KeyLen + MA.KeyLen$.

9.1.2 Encryption

The algorithm *DEM1.Encrypt* takes as input a secret key K , a label L , and a plaintext M . It runs as follows.

- a) Parse K as $K = k \| k'$, where k and k' are octet strings such that $|k| = SC.KeyLen$ and $|k'| = MA.KeyLen$.

- b) Compute $c = SC.Encrypt(k, M)$.
- c) Let $T = c \parallel L \parallel I2OSP(8 \cdot |L|, 8)$.
- d) Compute $MAC = MA.eval(k', T)$.
- e) Set $C_1 = c \parallel MAC$.
- f) Output C_1 .

9.1.3 Decryption

The algorithm *DEM1.Decrypt* takes as input a secret key K , a label L , and a ciphertext C_1 . It runs as follows.

- a) Parse K as $K = k \parallel k'$, where k and k' are octet strings such that $|k| = SC.KeyLen$ and $|k'| = MA.KeyLen$.
- b) If $|C_1| < MA.MACLen$, then **fail**.
- c) Parse C_1 as $C_1 = c \parallel MAC$, where c and MAC are octet strings such that $|MAC| = MA.MACLen$.
- d) Let $T = c \parallel L \parallel I2OSP(8 \cdot |L|, 8)$.
- e) Compute $MAC' = MA.eval(k', T)$.
- f) If $MAC \neq MAC'$, then **fail**.
- g) Compute $M = SC.Decrypt(k, c)$.
- h) Output M .

NOTE A detailed discussion of the security of this construction is found in Annex B.6.1. It is only remarked here that provided the underlying *SC* and *MA* satisfy the appropriate security requirements, then so too will *DEM1*.

9.2 DEM2

9.2.1 System parameters

DEM2 is a family of fixed-label-length data encapsulation mechanisms, parameterized by the following system parameters:

- *SC*: a symmetric cipher, as described in Clause 6.5;
- *MA*: a MAC algorithm, as described in Clause 6.3;
- *LabelLen*: a non-negative integer.

The value of *DEM2.LabelLen* is defined to be equal to the value of the system parameter *LabelLen*.

The value of *DEM2.KeyLen* is defined as $DEM2.KeyLen = SC.KeyLen + MA.KeyLen$.

9.2.2 Encryption

The algorithm *DEM2.Encrypt* takes as input a secret key K , a label L of length $LabelLen$, and a plaintext M . It runs as follows.

- a) Parse K as $K = k \parallel k'$, where k and k' are octet strings such that $|k| = SC.KeyLen$ and $|k'| = MA.KeyLen$.
- b) Compute $c = SC.Encrypt(k, M)$.
- c) Let $T = c \parallel L$.
- d) Compute $MAC = MA.eval(k', T)$.
- e) Set $C_1 = c \parallel MAC$.
- f) Output C_1 .

9.2.3 Decryption

The algorithm *DEM2.Decrypt* takes as input a secret key K , a label L of length $LabelLen$, and a ciphertext C_1 . It runs as follows.

- a) Parse K as $K = k \parallel k'$, where k and k' are octet strings such that $|k| = SC.KeyLen$ and $|k'| = MA.KeyLen$.
- b) If $|C_1| < MA.MACLen$, then **fail**.
- c) Parse C_1 as $C_1 = c \parallel MAC$, where c and MAC are octet strings such that $|MAC| = MA.MACLen$.
- d) Let $T = c \parallel L$.
- e) Compute $MAC' = MA.eval(k', T)$.
- f) If $MAC \neq MAC'$, then **fail**.
- g) Compute $M = SC.Decrypt(k, c)$.
- h) Output M .

NOTE 1 A detailed discussion of the security of this construction is found in Annex B.6.1. It is only remarked here that provided the underlying SC and MA satisfy the appropriate security requirements, then so too will *DEM2*.

NOTE 2 *DEM2* is provided mainly for compatibility with other standards.

9.3 DEM3

9.3.1 System parameters

DEM3 is a family of fixed-plaintext-length data encapsulation mechanisms, parameterized by the following system parameters:

- *MA*: a MAC algorithm, as described in Clause 6.3;
- *MsgLen*: a positive integer.

The value of *DEM3.MsgLen* is defined to be equal to the value of the system parameter *MsgLen*.

The value of *DEM3.KeyLen* is defined as $DEM3.KeyLen = MsgLen + MA.KeyLen$.

9.3.2 Encryption

The algorithm *DEM3.Encrypt* takes as input a secret key *K*, a label *L*, and a plaintext *M* of length *MsgLen*. It runs as follows.

- a) Parse *K* as $K = k \parallel k'$, where *k* and *k'* are octet strings such that $|k| = MsgLen$ and $|k'| = MA.KeyLen$.
- b) Compute $c = k \oplus M$.
- c) Let $T = c \parallel L$.
- d) Compute $MAC = MA.eval(k', T)$.
- e) Set $C_1 = c \parallel MAC$.
- f) Output C_1 .

9.3.3 Decryption

The algorithm *DEM3.Decrypt* takes as input a secret key *K*, a label *L*, and a ciphertext C_1 . It runs as follows.

- a) Parse *K* as $K = k \parallel k'$, where *k* and *k'* are octet strings such that $|k| = MsgLen$ and $|k'| = MA.KeyLen$.
- b) If $|C_1| \neq MsgLen + MA.MACLen$, then **fail**.
- c) Parse C_1 as $C_1 = c \parallel MAC$, where *c* and *MAC* are octet strings such that $|c| = MsgLen$ and $|MAC| = MA.MACLen$.
- d) Let $T = c \parallel L$.
- e) Compute $MAC' = MA.eval(k', T)$.
- f) If $MAC \neq MAC'$, then **fail**.
- g) Compute $M = k \oplus c$.
- h) Output *M*.

NOTE 1 A detailed discussion of the security of this construction is found in Annex B.6.1. It is only remarked here that provided the underlying *MA* satisfies the appropriate security requirement, then so too will *DEM3*.

NOTE 2 *DEM3* is provided mainly for compatibility with other standards.

10 ElGamal-based key encapsulation mechanisms

This clause describes several key encapsulation mechanisms based on the discrete logarithm problem:

- *ECIES-KEM* is described in Clause 10.2;
- *PSEC-KEM* is described in Clause 10.3;
- *ACE-KEM* is described in Clause 10.4.

NOTE All of these schemes are variations on the original ElGamal encryption scheme [18].

10.1 Concrete groups

ElGamal encryption is based on arithmetic in a finite group. For the purposes of describing key encapsulation mechanisms based on ElGamal encryption, a group is described as an abstract data type. The description and analysis of these schemes relies on this abstract interface; however, this part of ISO/IEC 18033 only allows an implementation to use certain types of groups when instantiating this abstract data type.

As a matter of convention, additive notation will always be used for a group. Also, group elements will be typeset in boldface, and **0** denotes the identity element of the group.

A *concrete group* Γ 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 $\mathcal{H}$.
- $\mathbf{g}$ is a generator for $\mathcal{G}$.
- μ is the order (i.e., size) of $\mathcal{G}$, and ν is the index of $\mathcal{G}$ in $\mathcal{H}$, i.e., $\nu = |\mathcal{H}|/\mu$.

It is required that μ is prime. For some cryptographic schemes, it is further required that $\gcd(\mu, \nu) = 1$.

- $\mathcal{E}(\mathbf{a}, \textit{fmt})$ is an “encoding” function that maps a group element $\mathbf{a} \in \mathcal{H}$ to an octet string.

The second argument *fmt* is a format specifier that is used to choose from one of a small number of several possible formats for the encoding of a group element. The allowable values of *fmt* depend on the group.

The following requirements shall be met:

- The set of all outputs of $\mathcal{E}$ is prefix free.
- The identity element has a unique encoding; that is, for all format specifiers *fmt*, *fmt'*, we have $\mathcal{E}(\mathbf{0}, \textit{fmt}) = \mathcal{E}(\mathbf{0}, \textit{fmt}')$.

- Except on the identity element, the encoding function is one to one; that is, for all $\mathbf{a}, \mathbf{a}' \in \mathcal{H}$ and for all format specifiers fmt, fmt' , if $(\mathbf{a}, fmt) \neq (\mathbf{a}', fmt')$, and if either $\mathbf{a} \neq \mathbf{0}$ or $\mathbf{a}' \neq \mathbf{0}$, then $\mathcal{E}(\mathbf{a}, fmt) \neq \mathcal{E}(\mathbf{a}', fmt')$.

An octet string x is called a *valid encoding* of a group element $\mathbf{a} \in \mathcal{H}$ if $x = \mathcal{E}(\mathbf{a}, fmt)$ for some format specifier fmt .

- $\mathcal{D}(x)$ is the function that **fails** if x is not a valid encoding of an element of $\mathcal{H}$; otherwise, it returns the unique group element $\mathbf{a} \in \mathcal{H}$ such that $\mathcal{E}(\mathbf{a}, fmt) = x$ for some format specifier fmt .
- $\mathcal{E}'(\mathbf{a})$ is a “partial encoding” function that maps a group element $\mathbf{a} \in \mathcal{H}$ to an octet string.

It is required that the set of all outputs of $\mathcal{E}'$ is prefix free.

An octet string x is called a *valid partial encoding* of a group element $\mathbf{a}$ if $x = \mathcal{E}'(\mathbf{a})$.

- $\mathcal{D}'(x)$ is a function that either **fails** if x is not a valid partial encoding of an element of $\mathcal{H}$; otherwise, it returns the set containing all group elements $\mathbf{a} \in \mathcal{H}$ such that $\mathcal{E}'(\mathbf{a}) = x$. It is assumed that the size of this set is bounded by a small constant.

It is assumed that arithmetic in $\mathcal{H}$ can be carried out efficiently. Also, 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.

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

A set $\{\mathcal{E}(\mathbf{a}_1, fmt_1), \dots, \mathcal{E}(\mathbf{a}_m, fmt_m)\}$ of valid encodings of group elements is called *consistent* if the encodings of all non-identity group elements use the same format specifier; that is, for all $1 \leq i, j \leq m$, if $\mathbf{a}_i \neq \mathbf{0}$ and $\mathbf{a}_j \neq \mathbf{0}$, then $fmt_i = fmt_j$. Given the above assumptions, one can efficiently test if a given set of valid encodings is consistent.

NOTE Different cryptographic applications will make different intractability assumptions about a group. These assumptions are discussed in Annex B.8.

10.1.1 Allowable concrete groups

This part of ISO/IEC 18033 allows only the following two families of concrete groups, described below in Clauses 10.1.2 and 10.1.3.

10.1.2 Subgroups of explicitly given finite fields

Let F be an explicitly given finite field, as defined in Clause 5.3, and consider the multiplicative group F^* of units in F . Let $\mathcal{H}$ denote F^* . Let $\mathcal{G}$ denote any prime-order subgroup of F^* , and let $\mathbf{g}$ be a generator for $\mathcal{G}$. Set $\mu = |\mathcal{G}|$ and $\nu = (|F| - 1)/\mu$.

Because $\mathcal{H}$ is itself cyclic, it follows that $\mathcal{G}$ contains all elements of $\mathcal{H}$ whose order divides μ , even if $\gcd(\mu, \nu) \neq 1$. Thus, one may always test if an element $\mathbf{a} \in \mathcal{H}$ lies in $\mathcal{G}$ by testing if $\mu \cdot \mathbf{a} = \mathbf{0}$; there may, however, be other, more efficient tests; for example, if F is a prime finite field, and $\nu = 2$, this test may be implemented via a Jacobi symbol computation.

The encoding map $\mathcal{E}$ is implemented using the function $FE2OSP_F$, so that all group elements are encoded as octet strings of length $\lceil \log_{256} |F| \rceil$. Only one format is allowed. The map $\mathcal{D}$ is implemented using $OS2FEP_F$, and **fails** if $OS2FEP_F$ **fails** or yields 0_F . The function $\mathcal{E}'$ is the same as $\mathcal{E}$, and $\mathcal{D}'$ is the same as $\mathcal{D}$.

10.1.3 Subgroups of Elliptic Curves

Let E be an elliptic curve defined over an explicitly given finite field F , as in Clause 5.4. Let $\mathcal{H}$ denote this group E . Let $\mathcal{G}$ denote a prime-order subgroup of $\mathcal{H}$, and let $\mathbf{g}$ be a generator for $\mathcal{G}$. Let μ be the order of $\mathcal{G}$, and ν be its index in $\mathcal{H}$.

Observe that $\mathcal{H}$ is not in general cyclic. If $\gcd(\mu, \nu) = 1$, then one may test if an element $\mathbf{a} \in \mathcal{H}$ lies in $\mathcal{G}$ by testing if $\mu \cdot \mathbf{a} = \mathbf{0}$. If $\gcd(\mu, \nu) \neq 1$, then more information about the group structure of E is required in order to construct an efficient test for membership in $\mathcal{G}$.

The encoding/decoding maps $\mathcal{E}$ and $\mathcal{D}$ are implemented using the functions $EC2OSP_E$ and $OS2ECP_E$. Thus, the encoding of a point is an octet string of length either 1, $1 + \lceil \log_{256} |F| \rceil$, or $1 + 2\lceil \log_{256} |F| \rceil$. The set of allowable format specifiers may be chosen to be any non-empty subset of $\{\text{uncompressed, compressed, hybrid}\}$. Thus, a concrete group defined using an elliptic curve may, but need not, allow multiple encoding formats.

The partial encoding map $\mathcal{E}'$ is defined as follows. Given a point P on E , if $P = \mathcal{O}$, then the output is $FE2OSP_F(0_F)$, and if $P = (x, y) \neq \mathcal{O}$, where $x, y \in F$, then the output is $FE2OSP_F(x)$. Thus, the output of $\mathcal{E}'$ is an octet string of length $\lceil \log_{256} |F| \rceil$.

10.2 ECIES-KEM

This clause describes the key encapsulation mechanism *ECIES-KEM*.

NOTE *ECIES-KEM* is based on the work of Abdalla, Bellare, and Rogaway [1, 2].

10.2.1 System parameters

ECIES-KEM is a family of key encapsulation mechanisms, parameterized by the following system parameters:

- Γ : a concrete group

$$\Gamma = (\mathcal{H}, \mathcal{G}, \mathbf{g}, \mu, \nu, \mathcal{E}, \mathcal{D}, \mathcal{E}', \mathcal{D}'),$$
 as described in Clause 10.1;
- *KDF*: a key derivation function, as described in Clause 6.2;
- *CofactorMode*: one of two values: 0 or 1.
- *OldCofactorMode*: one of two values: 0 or 1.
- *CheckMode*: one of two values: 0 or 1.
- *SingleHashMode*: one of two values: 0 or 1.
- *KeyLen*: a positive integer.

Any combination of system parameters is allowed, except for the following restrictions:

- At most one of *CofactorMode*, *OldCofactorMode*, and *CheckMode* may be 1.
- If $\nu > 1$ and *CheckMode* = 0, then we must have $\gcd(\mu, \nu) = 1$.

The value of *ECIES-KEM.KeyLen* is defined to be equal to the value of the system parameter *KeyLen*.

NOTE The values of *CofactorMode* and *CheckMode* are used only by the decryption algorithm.

10.2.2 Key generation

The key generation algorithm *ECIES-KEM.KeyGen* takes no input, and runs as follows.

- a) Generate a random number $x \in [1 \dots \mu)$.
- b) Compute $\mathbf{h} = x \cdot \mathbf{g}$.
- c) Output the public key:
 - $\mathbf{h}$: a non-zero element of $\mathcal{G}$.
- d) Output the private key:
 - x : an integer in the set $[1 \dots \mu)$

10.2.3 Encryption

The encryption algorithm *ECIES-KEM.Encrypt* takes as input a public key, consisting of $\mathbf{h} \in \mathcal{G} \setminus \{\mathbf{0}\}$, together with an encryption option *fmt* that specifies the format to be used for encoding group elements. It runs as follows.

- a) Generate a random number $r \in [1 \dots \mu)$.
- b) If *OldCofactorMode* = 1, then set $r' = r \cdot \nu \bmod \mu$; otherwise, set $r' = r$.
- c) Compute $\tilde{\mathbf{g}} = r \cdot \mathbf{g}$ and $\tilde{\mathbf{h}} = r' \cdot \mathbf{h}$.
- d) Set $C_0 = \mathcal{E}(\tilde{\mathbf{g}}, \text{fmt})$.
- e) If *SingleHashMode* = 1, then let Z be the null octet string; otherwise, let $Z = C_0$.
- f) Set $PEH = \mathcal{E}'(\tilde{\mathbf{h}})$.
- g) Set $K = \text{KDF}(Z \parallel PEH, \text{KeyLen})$.
- h) Output the ciphertext C_0 and the secret key K .

10.2.4 Decryption

The decryption algorithm *ECIES-KEM.Decrypt* takes as input a private key, consisting of $x \in [1 \dots \mu)$, and a ciphertext C_0 . It runs as follows.

- a) Set $\tilde{\mathbf{g}} = \mathcal{D}(C_0)$; this step **fails** if C_0 is not a valid encoding of an element of $\mathcal{H}$.
- b) If $CheckMode = 1$, test if $\tilde{\mathbf{g}} \in \mathcal{G}$; if not, then **fail**.
- c) If $CofactorMode = 1$ or $OldCofactorMode = 1$, set $\hat{\mathbf{g}} = \nu \cdot \tilde{\mathbf{g}}$; otherwise, set $\hat{\mathbf{g}} = \tilde{\mathbf{g}}$.
- d) If $CofactorMode = 1$, then set $\hat{x} = \nu^{-1}x \bmod \mu$; otherwise, set $\hat{x} = x$.
- e) Compute $\tilde{\mathbf{h}} = \hat{x} \cdot \hat{\mathbf{g}}$.
- f) If $\tilde{\mathbf{h}} = \mathbf{0}$, then **fail**.
- g) If $SingleHashMode = 1$, then let Z be the null octet string; otherwise, let $Z = C_0$.
- h) Set $PEH = \mathcal{E}'(\tilde{\mathbf{h}})$.
- i) Set $K = KDF(Z \parallel PEH, KeyLen)$.
- j) Output the secret key K .

NOTE 1 Using $CofactorMode = 1$ or $OldCofactorMode = 1$ may yield a significant performance benefit if ν is fairly small. An advantage of using $CofactorMode = 1$ is that the behavior of the encryption algorithm is not affected by the value of $CofactorMode$.

NOTE 2 When using $CofactorMode = 1$, an implementation could simply pre-compute and store the value $\hat{x}$, instead of the value x .

NOTE 3 When using $SingleHashMode = 1$, even if Γ supports multiple encoding formats, the value of fnt used during encryption does not affect any of the computations, except for the format of the resulting ciphertext. Thus, given a ciphertext C_0 that is an encoding of a group element $\tilde{\mathbf{g}}$, any ciphertext C'_0 that is also an encoding of $\tilde{\mathbf{g}}$ will decrypt in the same way as C_0 .

NOTE 4 A discussion of the security of this scheme can be found in Annex B.9.

10.3 PSEC-KEM

This clause describes the key encapsulation mechanism *PSEC-KEM*.

NOTE *PSEC-KEM* is based on the work of Fujisaki and Okamoto [26].

10.3.1 System parameters

PSEC-KEM is a family of key encapsulation mechanisms, parameterized by the following system parameters:

- Γ : a concrete group

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

 as described in Clause 10.1;
- KDF : a key derivation function, as described in Clause 6.2;
- $SeedLen$: a positive integer;
- $KeyLen$: a positive integer.

10.3.2 Key Generation

The key generation algorithm *PSEC-KEM.KeyGen* takes no input, and runs as follows.

- a) Generate a random number $x \in [0 \dots \mu)$.
- b) Compute $\mathbf{h} = x \cdot \mathbf{g}$.
- c) Output the public key:
 - $\mathbf{h}$: an element of $\mathcal{G}$.
- d) Output the private key:
 - x : an integer in the set $[0 \dots \mu)$.

10.3.3 Encryption

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

The encryption algorithm *PSEC-KEM.Encrypt* takes as input a public key, consisting of $\mathbf{h} \in \mathcal{G}$, together with an encryption option *fmt* that specifies the format to be used for encoding group elements. It runs as follows.

- a) Generate a random octet string *seed* of length *SeedLen*.
- b) Compute

$$t = KDF(I0 \parallel \textit{seed}, \lceil \log_{256} \mu \rceil + 16 + \textit{KeyLen}),$$
 an octet string of length $\lceil \log_{256} \mu \rceil + 16 + \textit{KeyLen}$.
- c) Parse t as $t = u \parallel K$, where u and K are octet strings such that $|u| = \lceil \log_{256} \mu \rceil + 16$ and $|K| = \textit{KeyLen}$.
- d) Compute $r = OS2IP(u) \bmod \mu$.
- e) Compute $\tilde{\mathbf{g}} = r \cdot \mathbf{g}$ and $\tilde{\mathbf{h}} = r \cdot \mathbf{h}$.
- f) Set $EG = \mathcal{E}(\tilde{\mathbf{g}}, \textit{fmt})$ and $PEH = \mathcal{E}'(\tilde{\mathbf{h}})$.
- g) Set $\textit{SeedMask} = KDF(I1 \parallel EG \parallel PEH, \textit{SeedLen})$.
- h) Set $\textit{MaskedSeed} = \textit{seed} \oplus \textit{SeedMask}$.
- i) Set $C_0 = EG \parallel \textit{MaskedSeed}$.
- j) Output the secret key K and the ciphertext C_0 .

10.3.4 Decryption

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

The decryption algorithm *PSEC-KEM.Decrypt* takes as input a private key, consisting of $x \in [0 \dots \mu)$, and a ciphertext C_0 . It runs as follows.

- a) Parse C_0 as $C_0 = EG \parallel MaskedSeed$, EG and $MaskedSeed$ are octet strings such that $|MaskedSeed| = SeedLen$; this step **fails** if $|C_0| < SeedLen$.
- b) Set $\tilde{\mathbf{g}} = \mathcal{D}(EG)$; this step fails if EG is not a valid encoding of a group element.
- c) Compute $\tilde{\mathbf{h}} = x \cdot \tilde{\mathbf{g}}$.
- d) Set $PEH = \mathcal{E}'(\tilde{\mathbf{h}})$.
- e) Set $SeedMask = KDF(I1 \parallel EG \parallel PEH, SeedLen)$.
- f) Set $seed = MaskedSeed \oplus SeedMask$.
- g) Compute

$$t = KDF(I0 \parallel seed, \lceil \log_{256} \mu \rceil + 16 + KeyLen),$$
 an octet string of length $\lceil \log_{256} \mu \rceil + 16 + KeyLen$.
- h) Parse t as $t = u \parallel K$, where u and K are octet strings such that $|u| = \lceil \log_{256} \mu \rceil + 16$ and $|K| = KeyLen$.
- i) Compute $r = OS2IP(u) \bmod \mu$.
- j) Compute $\bar{\mathbf{g}} = r \cdot \mathbf{g}$.
- k) Test if $\bar{\mathbf{g}} = \tilde{\mathbf{g}}$; if not, then **fail**.
- l) Output the secret key K .

NOTE A discussion of the security of this scheme can be found in Annex B.10.

10.4 ACE-KEM

This clause describes the key encapsulation mechanism *ACE-KEM*.

NOTE *ACE-KEM* is based on the work of Cramer and Shoup [13, 14].

10.4.1 System parameters

ACE-KEM is a family of key encapsulation mechanisms, parameterized by the following system parameters:

- Γ : a concrete group

$$\Gamma = (\mathcal{H}, \mathcal{G}, \mathbf{g}, \mu, \nu, \mathcal{E}, \mathcal{D}, \mathcal{E}', \mathcal{D}'),$$
 as described in Clause 10.1;
- KDF : a key derivation function, as described in Clause 6.2;
- $Hash$: a cryptographic hash function, as described in Clause 6.1;
- $CofactorMode$: one of two values: 0 or 1.
- $KeyLen$: a positive integer.