# PSEC-KEM Specification
# version 2.2

NTT Information Sharing Platform Laboratories,
NTT Corporation

April 14, 2008

## History

| | | |
|---|---|---|
| version 2.2 | Apr. 14, 2008 | Fixed conversions in Section 3, revised paragraphs in sections, 1, 3, and 5, and cleaned up the document. |
| version 2.1 | Jan. 18, 2008 | Revised appendices and corrected typos. |
| version 2.01 | Sep. 3, 2007 | Added SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512 in the allowable hash functions. |
| version 2.0 | Jun 14, 2007 | Revised as compatible with ISO/IEC 18033-2. |
| version 1.1 | May 14, 2002 | Fixed conversions in Section 3, and corrected typos. |
| version 1.0 | Sep. 26, 2001 | |

# Contents

# 1 Introduction

This document specifies key encapsulation mechanism PSEC-KEM.

A key encapsulation mechanism (KEM) is asymmetric cryptographic techniques to encrypt a secret key that can be used to encrypt an actual message using symmetric cryptographic techniques, called a data encapsulation mechanism (DEM). These techniques combined are called the KEM-DEM framework for realizing an asymmetric (hybrid) cipher. See Clause 8 in ISO/IEC 18033-2 [1] for definitions and usages of these terms.

The implementations based on this document are compatible with the counterparts of PSEC-KEM specified in ISO/IEC 18033-2 [1].

# 2 Notations

## 2.1 Notations

| | |
|---|---|
| bit | one of the two symbols '0' or '1'. |
| bit string | an ordered sequence of bits. |
| octet | a bit string of length 8. |
| octet string | an ordered sequence of octets. |
| $\mathbb{R}$ | the set of real numbers. |
| $\mathbb{Z}$ | the set of integers. |
| $\mathbb{N}$ | the set of positive integers. |
| $a := b$ | an operation of assigning $b$ to $a$. |
| $\mathbb{F}_{q^m}$ | a finite field with $q^m$ elements, where $q$ is a prime. |
| $\mathcal{O}$ | a point at infinity on an elliptic curve. |
| $\|$ | a concatenation operator for two bit strings or for two octet strings. This operator is often omitted for simplicity. |
| $\oplus$ | the bit-wise exclusive-or operation. |
| $\lceil y \rceil$ | for $y \in \mathbb{R}$, the least integer greater than or equal to $y$. |
| $\lfloor y \rfloor$ | for $y \in \mathbb{R}$, the greatest integer less than or equal to $y$. |
| $a \mid b$ | relation between integers $a$ and $b$ that holds if and only if $a$ divides $b$, i.e., there exists an integer $c$ such that $b = ac$. |
| $a \bmod m$ | for $a \in \mathbb{Z}$, $m \in \mathbb{N}$, the least non-negative integer $b$ that satisfies $m \mid (a - b)$. |
| $a^{-1} \bmod m$ | for $a \in \mathbb{Z}$, $m \in \mathbb{N}$, the least non-negative integer $b$ that satisfies $ab \bmod m = 1$. |

## 2.2 Elliptic Curve Parameter

An elliptic curve parameter $E$ in this document is specified by the following 9-tuple $(q, m, f(\boldsymbol{\beta}), \boldsymbol{a}, \boldsymbol{b}, P, p, pLen, qmLen)$, where the components have the following meanings:

- $q$, a prime number, such that $q \neq 3$;
- $m$, a positive integer;
- $f(\boldsymbol{\beta})$, a monic irreducible polynomial of degree $m$ over $\mathbb{F}_q$;
- $\boldsymbol{a}$, an element in $\mathbb{F}_{q^m}$;
- $\boldsymbol{b}$, an element in $\mathbb{F}_{q^m}$;
- $P$, a point $(x, y)$ on an elliptic curve, where
  - $\boldsymbol{x}$, an element in $\mathbb{F}_{q^m}$, and

– $\boldsymbol{y}$, an element in $\mathbb{F}_{q^m}$,

such that

– $\boldsymbol{y}^2 = \boldsymbol{x}^3 + \boldsymbol{a}\boldsymbol{x} + \boldsymbol{b}$ and $4\boldsymbol{a}^3 + 27\boldsymbol{b}^2 \neq 0$ if $q > 3$,
– $\boldsymbol{y}^2 + \boldsymbol{x}\boldsymbol{y} = \boldsymbol{x}^3 + \boldsymbol{a}\boldsymbol{x}^2 + \boldsymbol{b}$ and $\boldsymbol{b} \neq 0$ if $q = 2$;

- $p$, the order of $P$, a prime number;
- $pLen$, the value of $\lceil \log_{256} p \rceil$;
- $qmLen$, the value of $\lceil \log_{256} q^m \rceil$.

# 3 Data types and conversions

The scheme specified in this document involves operations using several different data types. Five data types are employed in this document: non-negative integers (I), field elements (FE), octet strings (OS), bit strings (BS), and elliptic curve points (ECP). The classification is meant to be abstract; throughout this document we make clear distinction between the five notions. So, for example, an octet string is regarded as distinct from a bit string.

In the following we describe how to convert one data type into another. A conversion function takes data represented in one of the five types, sometimes accepting an auxiliary input, and then outputs the data represented in a different type. Figure 1 illustrates which conversion function transforms which data type into another.

A conversion function always rejects unexpected input values by outputting a special symbol `INVALID`. The set of valid inputs is a subset of the set of the valid data type and is explicitly defined by the function. Several conversions described below are realized by composition of other conversions. When handling such a composition of conversion functions, we demand that the entire function should output `INVALID`, whenever one of the component functions outputs `INVALID`.

## 3.1 Integer-to-BitString Conversion (I2BSP)

Integers should be converted to bit strings as described in this section. Formally, the conversion routine, I2BSP$(x, l)$, is specified as follows:

| | | |
|---|---|---|
| **Input:** | $x$ | a non-negative integer |
| | $l$ | the bit length of the output, a non-negative integer |
| **Output:** | $B$ | a bit string of length $l$ |
| **Error:** | `INVALID` | |
| **Steps:** | | |

1. If $x \geq 2^l$, output `INVALID` and stop.
2. If $l = 0$, output the empty bit string and stop.
3. Represent $x$ in binary such that
$$x = x_{l-1}2^{l-1} + x_{l-2}2^{l-2} + \cdots + x_1 2 + x_0,$$
where $x_i \in \{0, 1\}$.
4. Set binary string $B := B_0 B_1 \cdots B_{l-1}$, such that $B_i := x_{l-1-i}$ for $0 \leq i \leq l-1$.
5. Output $B$.

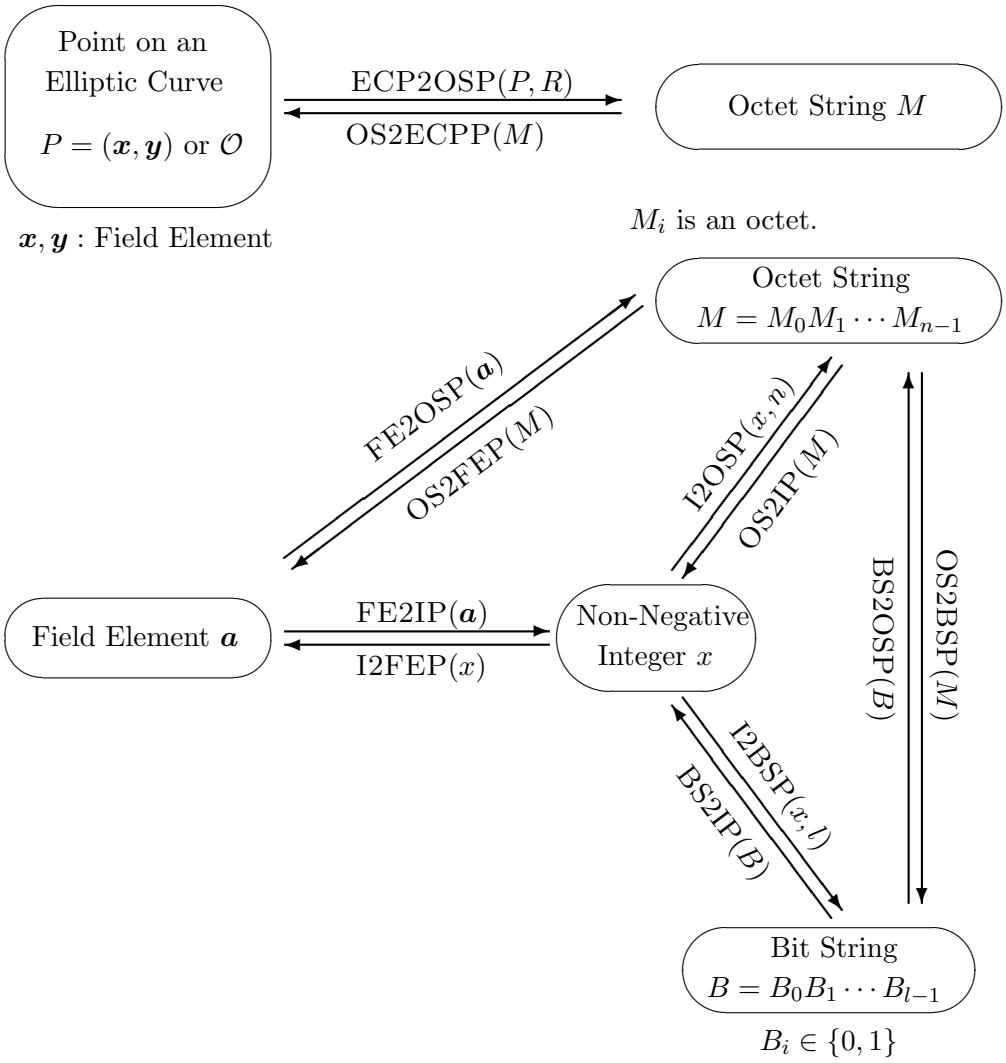


Figure 1: Conversions between data types

### 3.2 BitString-to-Integer Conversion (BS2IP)

Bit strings should be converted to integers as described in this section. Informally, the idea is simply to view the bit string as the binary representation of the integer. Formally, the conversion routine, BS2IP$(B)$, is specified as follows:

**Input:** $B$ a bit string of length $l$
**Output:** $x$ a non-negative integer
**Steps:**

Convert $B = B_0 B_1 \cdots B_{l-1}$ to an integer $x$ as follows:

1. If $l = 0$, output 0 and stop.
2. View each $B_i$ as an integer in $\{0, 1\}$, set $x_i := B_i$ for $0 \leq i \leq l - 1$, and compute

$$x := \sum_{i=0}^{l-1} 2^{(l-1-i)} x_i.$$

3. Output $x$.

### 3.3 BitString-to-OctetString Conversion (BS2OSP)

Bit strings should be converted to octet strings as described in this section. Informally, the idea is to pad the bit string with 0's on the left to make its length a multiple of 8, then chop the result up into octets. Formally, the conversion routine, BS2OSP($B$), is specified as follows:

**Input:** $B$ a bit string of length $l$
**Output:** $M$ an octet string of length $n = \left\lceil \frac{l}{8} \right\rceil$
**Steps:**

Convert the bit string $B = B_0 B_1 \cdots B_{l-1}$ to an octet string $M = M_0 M_1 \cdots M_{n-1}$ as follows:

1. If $l = 0$, output the empty octet string and stop.
2. For $j \in \{0, \ldots, 8n-1\}$, let:
$$\widetilde{B}_j = \begin{cases} B_{j-(8n-l)} & \text{if } j \geq 8n-l, \\ 0 & \text{if } j < 8n-l. \end{cases}$$
3. For $i \in \{0, \ldots, n-1\}$, set:
$$M_i := \widetilde{B}_{8i} \widetilde{B}_{8i+1} \cdots \widetilde{B}_{8i+7}$$
4. Output $M$.

### 3.4 OctetString-to-BitString Conversion (OS2BSP)

Octet strings should be converted to bit strings as described in this section. Informally, the idea is simply to view the octet string as a bit string. Formally, the conversion routine, OS2BSP($M$), is specified as follows:

**Input:** $M$ an octet string of length $n$
**Output:** $B$ a bit string of length $l = 8n$
**Steps:**

Convert the octet string $M = M_0 M_1 \cdots M_{n-1}$ to a bit string $B = B_0 B_1 \cdots B_{l-1}$ as follows:

1. If $l = 0$, output the empty bit string and stop.
2. For $i \in \{0, \ldots, n-1\}, j \in \{0, \ldots, 7\}$, set $B_{8i+j} \in \{0, 1\}$ as
$$M_i = B_{8i} B_{8i+1} \cdots B_{8i+7}$$
3. Output $B$.

### 3.5 Integer-to-OctetString Conversion (I2OSP)

Integers should be converted to octet strings as described in this section. Informally, the idea is to represent the integer in binary and then convert the resulting bit string to an octet string. Formally, the conversion routine, I2OSP$(x, n)$, is specified as follows:

| | | |
|---|---|---|
| **Input:** | $x$ | a non-negative integer |
| | $n$ | the octet length of output. |
| **Output:** | $M$ | an octet string of length $n$ |
| **Error:** | `INVALID` | |
| **Steps:** | | |

1. Let $M :=$ BS2OSP(I2BSP$(x, 8n)$).
2. Output $M$.

### 3.6 OctetString-to-Integer Conversion (OS2IP)

Octet strings should be converted to integers as described in this section. Informally, the idea is to represent the octet string as a bit string and then view the resulting bit string as the binary representation of the integer. Formally, the conversion routine, OS2IP$(M)$, is specified as follows:

| | | |
|---|---|---|
| **Input:** | $M$ | an octet string of length $n$ |
| **Output:** | $x$ | a non-negative integer |
| **Steps:** | | |

1. Let $x :=$ BS2IP(OS2BSP$(M)$).
2. Output $x$.

### 3.7 FieldElement-to-Integer Conversion (FE2IP)

Field elements should be converted to integers as described in this section. A field element should be represented as a polynomial with integer coefficients, which can be represented as a sequence of the coefficients. Informally, the idea is simply to view the sequence of the coefficients as the radix-$q$ representation of the integer, where $q$ is the characteristic of the field. Formally, the conversion routine, FE2IP$(\boldsymbol{a})$, is specified as follows:

| | | |
|---|---|---|
| **System parameters:** | $\mathbb{F}_{q^m}$ | a finite field with $q^m$ elements where $q$ is a prime, and $m > 0$ is an integer |
| **Input:** | $\boldsymbol{a}$ | a field element in $\mathbb{F}_{q^m}$ |
| **Output:** | $x$ | an integer in $\{0, \ldots, q^m - 1\}$ |
| **Steps:** | | |

Convert field element $\boldsymbol{a}$ to integer $x$ as follows:

if $m = 1$:

Field element $\boldsymbol{a}$ must be represented as an integer in $\{0, \ldots, q - 1\}$.

1. Let $x := \boldsymbol{a}$.
2. Output $x$.

if $m > 1$:

Field element $\boldsymbol{a}$ must be represented as a polynomial of at most $(m-1)$-th degree with coefficients in $\{0, \ldots, q-1\}$. Let $\beta$ be the variable of the polynomial.

1. Determine the coefficients $a_i \in \{0, \ldots, q-1\}$ for $i \in \{0, \ldots, m-1\}$ that satisfy
$$\boldsymbol{a} = \sum_{i=0}^{m-1} a_i \beta^i.$$
2. Compute
$$x := \sum_{i=0}^{m-1} a_i q^i.$$
3: Output $x$.

### 3.8 Integer-to-FieldElement Conversion (I2FEP)

Integers should be converted to field elements as described in this section. A field element should be represented as a polynomial with integer coefficients, and it can be represented as a sequence of the coefficients. Informally, the idea is to represent the integer with radix-$q$ positional number system where $q$ is the characteristic of the field, and then convert the each digit to the each coefficient of the polynomial. Formally, the conversion routine, I2FEP($x$), is specified as follows:

| | | |
|---|---|---|
| **System parameters:** | $\mathbb{F}_{q^m}$ | a finite field with $q^m$ elements where $q$ is a prime, and $m > 0$ is an integer |
| **Input:** | $x$ | an integer in $\{0, \ldots, q^m - 1\}$ |
| **Output:** | $\boldsymbol{a}$ | a field element in $\mathbb{F}_{q^m}$ |
| **Steps:** | | |

Convert integer $x$ to field element $\boldsymbol{a}$ as follows:

if $m = 1$:

A field element of $\mathbb{F}_{q^m}$ must be represented as an integer in $\{0, \ldots, q-1\}$.

1. Let $\boldsymbol{a} := x$.
2. Output $\boldsymbol{a}$.

if $m > 1$:

A field element of $\mathbb{F}_{q^m}$ must be represented as a polynomial of at most $(m-1)$-th degree with coefficients in $\{0, \ldots, q-1\}$. Let $\beta$ be the variable of the polynomial.

1. Expand $x$ into it's radix $q$ representation $x_i \in \{0, \ldots, q-1\}$ for $i \in \{0, \ldots, m-1\}$ that satisfies
$$x = \sum_{i=0}^{m-1} x_i q^i.$$
2. Compute
$$\boldsymbol{a} := \sum_{i=0}^{m-1} x_i \beta^i.$$
3: Output $\boldsymbol{a}$.

### 3.9 FieldElement-to-OctetString Conversion (FE2OSP)

The conversion routine, FE2OSP($\boldsymbol{a}$), is specified as follows:

| | | |
|---|---|---|
| **System parameters:** | $\mathbb{F}_{q^m}$ | a finite field with $q^m$ elements where $q$ is a prime, and $m > 0$ is an integer |
| | $n$ | an integer equivalent to $\left\lceil \frac{m \log_2 q}{8} \right\rceil$ |
| **Input:** | $\boldsymbol{a}$ | a field element in $\mathbb{F}_{q^m}$ |
| **Output:** | $M$ | an octet string |
| **Steps:** | | |

1. Let
$$M := \text{I2OSP}(\text{FE2IP}(\boldsymbol{a}), n).$$
2. Output $M$.

Note: The system parameters should be equivalent in FE2IP and FE2OSP.

### 3.10 OctetString-to-FieldElement Conversion (OS2FEP)

The conversion routine, OS2FEP($M$), is specified as follows:

| | | |
|---|---|---|
| **System parameters:** | $\mathbb{F}_{q^m}$ | a finite field with $q^m$ elements where $q$ is a prime, and $m > 0$ is an integer |
| | $n$ | an integer equivalent to $\left\lceil \frac{m \log_2 q}{8} \right\rceil$ |
| **Input:** | $M$ | an octet string |
| **Output:** | $\boldsymbol{a}$ | a field element in $\mathbb{F}_{q^m}$ |
| **Error:** | `INVALID` | |
| **Steps:** | | |

1. Let
$$\boldsymbol{a} := \text{I2FEP}(\text{OS2IP}(M)).$$
2. Output $\boldsymbol{a}$.

Note: The system parameters should be equivalent in I2FEP and OS2FEP.

### 3.11 EllipticCurvePoint-to-OctetString Conversion (ECP2OSP)

Elliptic curve points should be converted to octet strings as described in this section. Informally the idea is that, if point compression is being used, the compressed $y$-coordinate is placed in the leftmost octet of the octet string along with an indication that point compression is on, and the $x$-coordinate is placed in the remainder of the octet string; otherwise if point compression is off, the leftmost octet indicates that point compression is off, and remainder of the octet string contains the $x$-coordinate followed by the $y$-coordinate. Formally, the conversion routine, ECP2OSP($P, R$), is specified as follows:

**System parameters:** $E$ an elliptic curve parameter
**Input:** $P$ a point on an elliptic curve over $\mathbb{F}_{q^m}$
$R$ Compressed, Uncompressed, or Hybrid
**Output:** $M$ an octet string of length $n$

$$\text{where } \begin{cases} n = 1 & \text{if } P = \mathcal{O}, \\ n = \left\lceil \frac{m \log_2 q}{8} \right\rceil + 1 & \text{if } P \neq \mathcal{O} \text{ and } R \text{ is Compressed}, \\ n = 2\left\lceil \frac{m \log_2 q}{8} \right\rceil + 1 & \text{if } P \neq \mathcal{O} \text{ and } R \text{ is Uncompressed or Hybrid.} \end{cases}$$

**Steps:**

Convert $P$ to an octet string $M = M_0 M_1 \cdots M_{n-1}$ as follows:

1. If $P = \mathcal{O}$, output $M := \text{I2OSP}(0, 1)$ .
2. If $P = (\boldsymbol{x}, \boldsymbol{y}) \neq \mathcal{O}$ and $R =$ Compressed, proceed as follows:
   2.1. Set octet string $X := \text{FE2OSP}(\boldsymbol{x})$.
   2.2. Derive from $\boldsymbol{y}$ a single bit $\tilde{y}$ as follows (this allows the $y$-coordinate to be represented compactly using a single bit):
      2.2.1. If $q$ is an odd number, set $\tilde{y} := 0$ if $\boldsymbol{y} = \mathbf{0}$, otherwise set $\tilde{y} := y_i \bmod 2$, where $\boldsymbol{y} = y_{m-1}\beta^{m-1} + \cdots + y_1\beta + y_0$, and $i$ is the smallest integer such that $y_i \neq 0$.
      2.2.2. If $q = 2$, set $\tilde{y} := 0$ if $\boldsymbol{x} = \mathbf{0}$, otherwise compute $\boldsymbol{z} = z_{m-1}\beta^{m-1} + \cdots + z_1\beta + z_0$ such that $\boldsymbol{z} = \boldsymbol{y}\boldsymbol{x}^{-1}$ and set $\tilde{y} := z_0$.
   2.3. If $\tilde{y} = 0$, assign the value $\text{I2OSP}(2, 1)$ to the single octet $L$. If $\tilde{y} = 1$, assign the value $\text{I2OSP}(3, 1)$ to the single octet $L$.
   2.4. Output $M := L \,\|\, X$.
3. If $P = (\boldsymbol{x}, \boldsymbol{y}) \neq \mathcal{O}$ and $R =$ Uncompressed, proceed as follows:
   3.1. Set octet string $X := \text{FE2OSP}(\boldsymbol{x})$.
   3.2. Set octet string $Y := \text{FE2OSP}(\boldsymbol{y})$.
   3.3. Output $M := \text{I2OSP}(4, 1) \,\|\, X \,\|\, Y$.
4. If $P = (\boldsymbol{x}, \boldsymbol{y}) \neq \mathcal{O}$ and $R =$ Hybrid, proceed as follows:
   4.1. Set octet string $X := \text{FE2OSP}(\boldsymbol{x})$.
   4.2. Set octet string $Y := \text{FE2OSP}(\boldsymbol{y})$.
   4.3. Derive from $\boldsymbol{y}$ a single bit $\tilde{y}$ as follows (this allows the $y$-coordinate to be represented compactly using a single bit):
      4.3.1. If $q$ is an odd number, set $\tilde{y} := 0$ if $\boldsymbol{y} = \mathbf{0}$, otherwise set $\tilde{y} := y_i \bmod 2$, where $\boldsymbol{y} = y_{m-1}\beta^{m-1} + \cdots + y_1\beta + y_0$, and $i$ is the smallest integer such that $y_i \neq 0$.
      4.3.2. If $q = 2$, set $\tilde{y} := 0$ if $\boldsymbol{x} = \mathbf{0}$, otherwise compute $\boldsymbol{z} = z_{m-1}\beta^{m-1} + \cdots + z_1\beta + z_0$ such that $\boldsymbol{z} = \boldsymbol{y}\boldsymbol{x}^{-1}$ and set $\tilde{y} := z_0$.
   4.4. If $\tilde{y} = 0$, assign the value $\text{I2OSP}(6, 1)$ to the single octet $L$. If $\tilde{y} = 1$, assign the value $\text{I2OSP}(7, 1)$ to the single octet $L$.
   4.5. Output $M := L \,\|\, X \,\|\, Y$.

### 3.12 OctetString-to-EllipticCurvePoint Conversion (OS2ECPP)

Octet strings should be converted to elliptic curve points as described in this section. Informally, the idea is that, if the octet string represents a compressed point, the compressed $y$-coordinate is recovered from the leftmost octet, the $x$-coordinate is recovered from the remainder of the octet string, and then the point compression process is reversed; otherwise the leftmost octet of the octet string is removed, the $x$-coordinate is recovered from the left half of the remaining octet string, and the $y$-coordinate is recovered from the right half of the remaining octet string. Formally, the conversion routine, OS2ECPP($M$), is specified as follows:

| | | |
|---|---|---|
| **System parameters:** | $E$ | an elliptic curve parameter |
| **Input:** | $M$ | an octet string that is either<br>the single octet I2OSP$(0,1)$,<br>an octet string of length $n = \left\lceil \frac{m \log_2 q}{8} \right\rceil + 1$, or<br>an octet string of length $n = 2\left\lceil \frac{m \log_2 q}{8} \right\rceil + 1$. |
| **Output:** | $P$ | an elliptic curve point |
| **Error:** | `INVALID` | |
| **Steps:** | | |

Convert $M$ to a point $P$ on $E$ as follows:

1. If $M = \text{I2OSP}(0,1)$, output $P := \mathcal{O}$.
2. If $M$ has length $\left\lceil \frac{m \log_2 q}{8} \right\rceil + 1$ octets, proceed as follows:
   - 2.1. Parse $M = L \,\|\, X$ as a single octet $L$ followed by $\left\lceil \frac{m \log_2 q}{8} \right\rceil$ octets $X$.
   - 2.2. Set $\boldsymbol{x} := \text{OS2FEP}(X)$.
   - 2.3. If $L = \text{I2OSP}(2,1)$, set $\tilde{y} := 0$, and if $L = \text{I2OSP}(3,1)$, set $\tilde{y} := 1$. Otherwise output `INVALID` and stop.
   - 2.4. Derive from $\boldsymbol{x}$ and $\tilde{y}$ elliptic curve point $P := (\boldsymbol{x}, \boldsymbol{y})$, where:
     - 2.4.1. If $q$ is an odd number, compute the field element $\boldsymbol{w} := \boldsymbol{x}^3 + \boldsymbol{a}\boldsymbol{x} + \boldsymbol{b}$, and compute a square root $\boldsymbol{\gamma}$ of $\boldsymbol{w}$ in $\mathbb{F}_{q^m}$. Output `INVALID` and stop if there are no square roots in $\mathbb{F}_{q^m}$. Otherwise set $\boldsymbol{y} = \boldsymbol{0}$ if $\boldsymbol{\gamma} = \boldsymbol{0}$, otherwise set $\boldsymbol{y} := \boldsymbol{\gamma}$ if $\gamma_i \equiv \tilde{y} \bmod 2$, and set $\boldsymbol{y} := -\boldsymbol{\gamma}$ if $\gamma_i \not\equiv \tilde{y} \bmod 2$, where $\boldsymbol{\gamma} = \gamma_{m-1}\beta^{m-1} + \cdots + \gamma_1\beta + \gamma_0$, and $i$ is the smallest integer such that $\gamma_i \neq 0$.
     - 2.4.2. If $q = 2$ and $\boldsymbol{x} = \boldsymbol{0}$, set $\boldsymbol{y} := \boldsymbol{b}^{2^{m-1}}$ in $\mathbb{F}_{q^m}$.
     - 2.4.3. If $q = 2$ and $\boldsymbol{x} \neq \boldsymbol{0}$, compute the field element $\boldsymbol{\gamma} := \boldsymbol{x} + \boldsymbol{a} + \boldsymbol{b}\boldsymbol{x}^{-2}$ in $\mathbb{F}_{q^m}$, and find an element $\boldsymbol{z} = z_{m-1}\beta^{m-1} + \cdots + z_1\beta + z_0$ such that $\boldsymbol{z}^2 + \boldsymbol{z} = \boldsymbol{\gamma}$ in $\mathbb{F}_{q^m}$. Output `INVALID` and stop if no such $\boldsymbol{z}$ exists, otherwise set $\boldsymbol{y} := \boldsymbol{x}\boldsymbol{z}$ in $\mathbb{F}_{q^m}$ if $z_0 = \tilde{y}$, and set $\boldsymbol{y} := \boldsymbol{x}(\boldsymbol{z} + \boldsymbol{1})$ in $\mathbb{F}_{q^m}$ if $z_0 \neq \tilde{y}$.
   - 2.5. Output $P := (\boldsymbol{x}, \boldsymbol{y})$.
3. If $M$ has length $2\left\lceil \frac{m \log_2 q}{8} \right\rceil + 1$ octets, proceed as follows:

3.1. Parse $M = L \,\|\, X \,\|\, Y$ as a single octet $L$ followed by $\left\lceil \frac{m \log_2 q}{8} \right\rceil$ octets $X$ followed by $\left\lceil \frac{m \log_2 q}{8} \right\rceil$ octets $Y$.

3.2. Unless $L$ is I2OSP$(4, 1)$, I2OSP$(6, 1)$ or I2OSP$(7, 1)$, output `INVALID` and stop.

3.3. Set $\boldsymbol{x} :=$ OS2FEP$(X)$.

3.4. Set $\boldsymbol{y} :=$ OS2FEP$(Y)$.

3.5. If $P := (\boldsymbol{x}, \boldsymbol{y})$ does not satisfy the defining equation of elliptic curve $E$, then output `INVALID` and stop.

3.6. Output $P := (\boldsymbol{x}, \boldsymbol{y})$.

### 3.13 Partial EllipticCurvePoint-to-OctetString Conversion (PECP2OSP)

The conversion routine, PECP2OSP$(P)$, is specified as follows:

| | | |
|---|---|---|
| **System parameters:** | $E$ | an elliptic curve parameter |
| **Input:** | $P$ | a point on an elliptic curve over $\mathbb{F}_{q^m}$ |
| **Output:** | $M$ | an octet string of length $n = \left\lceil \frac{m \log_2 q}{8} \right\rceil$ |

**Steps:**

Convert $P$ to an octet string $M = M_0 M_1 \cdots M_{n-1}$ as follows:

1. If $P = \mathcal{O}$, output $M :=$ I2OSP$(0, n)$ .
2. If $P = (\boldsymbol{x}, \boldsymbol{y}) \neq \mathcal{O}$, output FE2OSP$(\boldsymbol{x})$ .

## 4 PSEC-KEM system parameters, and private and public keys

### 4.1 PSEC-KEM system parameters

PSEC-KEM is parameterized by the following system parameters:

- $E$, an elliptic curve parameter specified in Section 2.2;
- *KDF*, the key derivation function described in Section 6.2;
- *hLen*, a positive integer;
- *keyLen*, a positive integer.

### 4.2 PSEC-KEM private key

A PSEC-KEM private key is the following:

- $s$, a non-negative integer such that $0 \leq s < p$.

### 4.3 PSEC-KEM public key

A PSEC-KEM public key is the following:

- $W$, a point on $E$.

# 5 Key encapsulation mechanism PSEC-KEM

This section describes PSEC-KEM, which consists of the following three algorithms:

- KGP-PSEC, the key generation algorithm, takes no input and outputs a pair of public and private keys, $(W, s)$,
- ES-PSEC-KEM-Encrypt, the encryption operation, that takes as input public key $W$ and format $R$, and outputs a pair of a secret key and a ciphertext, $(k, c_0)$, and
- ES-PSEC-KEM-Decrypt, the decryption operation, that takes as input private key $s$ and ciphertext $c_0$, and outputs secret key $k$.

## 5.1 KGP-PSEC

KGP-PSEC() is defined as follows:

| | | |
|---|---|---|
| **Input:** | | KGP-PSEC takes no input |
| **Output:** | $W$ | the PSEC-KEM public key, a point on $E$ |
| | $s$ | the PSEC-KEM private key, a non-negative integer, $0 \leq s < p$ |

**Steps:**

1. Generate a random integer $s \in \{0, \cdots, p-1\}$.
2. Compute $W := sP$.
3. Output $W$ and $s$.

## 5.2 ES-PSEC-KEM

### 5.2.1 Encryption operation

ES-PSEC-KEM-Encrypt$(W, R)$ is defined as follows:

| | | |
|---|---|---|
| **Input:** | $W$ | the PSEC-KEM public key, a point on $E$ |
| | $R$ | Compressed, Uncompressed, or Hybrid |
| **Output:** | $k$ | an octet string |
| | $c_0$ | an octet string |

**Steps:**

1. Generate a random octet string, $r$, of length *hLen* (in octet).
2. Set $H := KDF(\mathrm{I2OSP}(0,4) \,\|\, r, pLen + 16 + keyLen)$.
3. Parse $H = t \,\|\, k$, such that $t$ is an octet string of length $pLen + 16$ (in octet) and $k$ is an octet string of length *keyLen* (in octet).
4. Compute $\alpha := \mathrm{OS2IP}(t) \bmod p$.
5. Compute $C_1 := \alpha P$.
6. Compute $Q := \alpha W$.
7. Set $c_2 := r \oplus KDF(\mathrm{I2OSP}(1,4) \,\|\, \mathrm{ECP2OSP}(C_1, R) \,\|\, \mathrm{PECP2OSP}(Q), hLen)$.
8. Set $c_0 := \mathrm{ECP2OSP}(C_1, R) \,\|\, c_2$.
9. Output $(k, c_0)$.

#### 5.2.2 Decryption operation

ES-PSEC-KEM-DECRYPT$(s, c_0)$ is defined as follows:

| | | |
|---|---|---|
| **Input:** | $s$ | the PSEC-KEM private key, a non-negative integer, $0 \leq s < p$ |
| | $c_0$ | an octet string |
| **Output:** | $k$ | an octet string |
| **Error:** | `INVALID` | |
| **Steps:** | | |

1. If the octet length of $c_0$ is less than *hLen*, output `INVALID` and stop.
2. Parse $c_0 = g \,\|\, c_2$, where $g$ and $c_2$ are octet strings such that the octet length of $c_2$ is *hLen*.
3. Compute $C_1 := \mathrm{OS2ECPP}(g)$. If OS2ECPP outputs `INVALID`, output `INVALID` and stop.
4. Compute $Q := sC_1$.
5. Set $r := c_2 \oplus \mathit{KDF}(\mathrm{I2OSP}(1, 4) \,\|\, g \,\|\, \mathrm{PECP2OSP}(Q), \mathit{hLen})$.
6. Set $h := \mathit{KDF}(\mathrm{I2OSP}(0, 4) \,\|\, r, \mathit{pLen} + 16 + \mathit{keyLen})$.
7. Parse $h = t \,\|\, k$, such that $t$ is an octet string of length $\mathit{pLen} + 16$ (in octet) and $k$ is an octet string of length *keyLen* (in octet).
8. Compute $\alpha := \mathrm{OS2IP}(t) \bmod p$.
9. Check $C_1 = \alpha P$. If it holds, output $k$. Otherwise, output `INVALID` and stop.

# 6 Auxiliary techniques

## 6.1 Hash functions

The allowable hash functions are SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512, described in ISO/IEC 10118-3 [2].

## 6.2 Key derivation functions

As a key derivation function, the function described in Section 6.2.1 is recommended. The function is referred to as KDF1 in ISO/IEC 18033-2 [1].

### 6.2.1 MGF1

MGF1 is a mask generation function, parameterized by a hash function, as described in Section 6.1. MGF1$(M, n)$ is defined as follows:

| | | |
|---|---|---|
| **System parameters:** | *Hash* | a hash function |
| | *hashLen* | the length in octets of the hash function output, a positive integer |
| **Input:** | $M$ | a seed from which a mask is generated, an octet string |
| | $n$ | the octet length of the output, a positive integer |
| **Output:** | *mask* | a mask, an octet string of length $n$ |
| **Error:** | `INVALID` | |
| **Steps:** | | |

1. Let $n_0$ be the octet length of $M$. If $n_0 + 4$ is greater than the input limitation for the hash function, output `INVALID` and stop.
2. Let $cThreshold := \left\lceil \frac{n}{hashLen} \right\rceil$.
3. If $cThreshold > 2^{32}$, output `INVALID` and stop.
4. Let $M'$ be the empty octet string.
5. Let $counter := 0$, i.e., set $counter$ as the integer zero.
   (a) Convert $counter$ to an octet string $C$ of length 4 octets:
   $$C := \mathrm{I2OSP}(counter, 4).$$
   (b) Concatenate $M$ and $C$, and apply the hash function to the result to produce a hash value $H$ of length $hashLen$ octets:
   $$H := Hash(M \,\|\, C).$$
   (c) Concatenate $M'$ and $H$ to the octet string $M'$:
   $$M' := M' \,\|\, H.$$
   (d) Let $counter := counter + 1$. If $counter < cThreshold$, go back to step 5a.
6. Let $mask$ be the leftmost $n$ octets of the octet string $M'$:
   $$mask := M'_0 M'_1 \cdots M'_{n-1}.$$
7. Output $mask$.

# References


[1] ISO/IEC 18033 – Information technology – Security techniques – Encryption algorithms – Part 2: Asymmetric ciphers, ISO, 2005a.

[2] ISO/IEC 10118 – Information technology – Security techniques – Hash functions – Part 3: Dedicated Hash functions, ISO, 2004.

# Appendix

## A Security requirements of parameters

The following conditions are required:

$$\begin{aligned} pLen &\geq 20 \\ hLen &\geq 16 \end{aligned}$$

## B Recommended parameters for secure implementation

The following conditions are recommended:

$$\begin{aligned} pLen &\geq 32 \quad (256bit) \\ KDF &= \text{MGF1(SHA-256}, hashLen = 32) \\ hLen &= 32 \\ R &= \text{Compressed} \\ keyLen &= 32 \quad (256bit) \end{aligned}$$

It is known that some classes of elliptic curves are vulnerable or susceptible to the known attacks in the security sense. To avoid an inappropriate choice of curves, see SECG (http://www.secg.org/).

## C ASN.1 Syntax

```
--########################################################################
ntt-ds OBJECT IDENTIFIER ::= {
  itu-t(0) networkoperator(3) ntt(4401) ds(5) }
id-PSEC-KEM-v2   OBJECT IDENTIFIER ::={ ntt-ds 3 1 8 }

DEFINITIONS EXPLICIT TAGS ::= BEGIN
-- EXPORTS All; --
IMPORTS
BlockAlgorithms
FROM EncryptionAlgorithms-3 { iso(1) standard(0)
encryption-algorithms(18033) part(3)
asn1-module(0) algorithm-object-identifiers(0) }
HashFunctionAlgs, id-sha1, NullParms
FROM DedicatedHashFunctions { iso(1) standard(0)
hash-functions(10118) part(3) asn1-module(1)
dedicated-hash-functions(0) };
--########################################################################
-- oid definitions
OID ::= OBJECT IDENTIFIER -- alias
-- Synonyms --
is18033-2 OID ::= { iso(1) standard(0) is18033(18033) part2(2)}
id-ac OID ::= { is18033-2 asymmetric-cipher(1) }
id-kem OID ::= { is18033-2 key-encapsulation-mechanism(2) }
```

```
id-dem OID ::= { is18033-2 data-encapsulation-mechanism(3) }
id-sc OID ::= { is18033-2 symmetric-cipher(4) }
id-kdf OID ::= { is18033-2 key-derivation-function(5) }
id-rem OID ::= { is18033-2 rsa-encoding-method(6) }
id-hem OID ::= { is18033-2 himer-encoding-method(7) }
id-ft OID ::= { is18033-2 field-type(8) }
-- Key encapsulation mechanisms --
id-kem-psec OID ::= { id-kem psec(2) }
-- Data encapsulation mechanisms --
id-dem-dem1 OID ::= { id-dem dem1(1) }
id-dem-dem2 OID ::= { id-dem dem2(2) }
id-dem-dem3 OID ::= { id-dem dem3(3) }
-- Symmetric ciphers --
id-sc-sc1 OID ::= { id-sc sc1(1) }
id-sc-sc2 OID ::= { id-sc sc2(2) }
-- Key derivation functions --
id-kdf-kdf1 OID ::= { id-kdf kdf1(1) }
id-kdf-kdf2 OID ::= { id-kdf kdf2(2) }
-- new field types oids
-- id-ft-prime-field OID ::= { id-ft prime-field(1) }
-- used only to define new basis type
id-ft-characteristic-two OID ::= { id-ft characteristic-two(2) }
id-ft-odd-characteristic OID ::= { id-ft odd-characteristic(3) }
id-ft-characteristic-two-basis OID ::=
{ id-ft-characteristic-two basisType(1) }
charTwoPolynomialBasis OID ::=
{ id-ft-characteristic-two-basis
charTwoPolynomialBasis(1) }
id-ft-odd-characteristic-basis OID ::= { id-ft-odd-characteristic
basisType(1)}
oddCharPolynomialBasis OID ::= {id-ft-odd-characteristic-basis
oddCharPolynomialBasis(1)}
-- MGF1 in PKCS #1 is equivalent to KDF1 here
-- id-mgf1 should be used instead of id-kdf-kdf1 for compatibility
-- with existing implementations
alg-mgf1-sha1 RsaesKeyDerivationFunction ::= {
algorithm id-mgf1,
parameters HashFunction : alg-sha1
}
alg-sha1 HashFunction ::= {
algorithm id-sha1,
parameters NullParms : NULL
}
pkcs-1 OID ::= {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) 1}
id-mgf1 OBJECT IDENTIFIER ::= {pkcs-1 8}
id-itu-sha1  OBJECT IDENTIFIER  ::=  { iso(1)
                          identified-organization(3) oiw(14)
                          secsig(3) algorithms(2) 26 }
id-itu-sha224  OBJECT IDENTIFIER  ::=  {{ joint-iso-itu-t(2)
```

```
                          country(16) us(840) organization(1) gov(101)
                          csor(3) nistalgorithm(4) hashalgs(2) 4 }
id-itu-sha256  OBJECT IDENTIFIER  ::=  { joint-iso-itu-t(2)
                          country(16) us(840) organization(1) gov(101)
                          csor(3) nistalgorithm(4) hashalgs(2) 1 }
id-itu-sha384  OBJECT IDENTIFIER  ::=  { joint-iso-itu-t(2)
                          country(16) us(840) organization(1) gov(101)
                          csor(3) nistalgorithm(4) hashalgs(2) 2 }
id-itu-sha512  OBJECT IDENTIFIER  ::=  { joint-iso-itu-t(2)
                          country(16) us(840) organization(1) gov(101)
                          csor(3) nistalgorithm(4) hashalgs(2) 3 }
id-camellia128-cbc OBJECT IDENTIFIER ::=
iso(1) member-body(2) 392 200011 61 security(1)
algorithm(1) symmetric-encryption-algorithm(1) camellia128-cbc(2)
id-aes OBJECT IDENTIFIER ::= { joint-iso-itu-t(2) country(16) us(840)
               organization(1) gov(101) csor(3)_ nistAlgorithms(4)  1 }
id-aes128-CBC OBJECT IDENTIFIER ::= { id-aes 2 }


--#####################################################################
-- KEM information objects
KeyEncapsulationMechanism ::= AlgorithmIdentifier {{ KEMAlgorithms }}
KEMAlgorithms ALGORITHM ::= {
{ OID id-kem-psec PARMS PsecKemParameters } |
{  OID id-PSEC-KEM-v2 PARMS PsecKemParameters },
... -- Expect additional algorithms --
}
--#####################################################################
-- PSEC-KEM
-- an element of the group given in PsecKemParameters (may be 0)
PsecKemPrivateKey ::= INTEGER
PsecKemPublicKey ::= FieldElement
PsecKemParameters ::= SEQUENCE {
group Group OPTIONAL,
keyDerivationFunction KeyDerivationFunction,
seedLength INTEGER (1..MAX),
keyLength KeyLength -- Length by byte
}
-- DEM specifications
DataEncapsulationMechanism ::= AlgorithmIdentifier {{DEMAlgorithms}}
DEMAlgorithms ALGORITHM ::= {
{ OID id-dem-dem1 PARMS Dem1Parameters } |
{ OID id-dem-dem2 PARMS Dem2Parameters } |
{ OID id-dem-dem3 PARMS Dem3Parameters },
... -- Expect additional algorithms --
}
Dem1Parameters ::= SEQUENCE{
symmetricCipher SymmetricCipher,
mac MacAlgorithm
```

```
}
Dem2Parameters ::= SEQUENCE{
symmetricCipher SymmetricCipher,
mac MacAlgorithm,
labelLength INTEGER (0..MAX)
}
Dem3Parameters ::= SEQUENCE{
mac MacAlgorithm,
msgLength INTEGER (0..MAX)
}
--#####################################################################
-- finite field, group, and elliptic curve representations
Group ::= CHOICE {
groupOid OBJECT IDENTIFIER,
groupHashId OCTET STRING, -- defined in RFC2528
groupParameters GroupParameters
}
GroupParameters ::= CHOICE {
explicitFiniteFieldSubgroup
[0] ExplicitFiniteFieldSubgroupParameters,
ellipticCurveSubgroup
[1] EllipticCurveSubgroupParameters
}
ExplicitFiniteFieldSubgroupParameters ::= SEQUENCE {
fieldID FieldID {{FieldTypes}},
generator FieldElement,
subgroupOrder INTEGER,
subgroupIndex INTEGER
}
FIELD-ID ::= TYPE-IDENTIFIER
FieldID { FIELD-ID:IOSet } ::= SEQUENCE {
fieldType FIELD-ID.&id({IOSet}),
parameters FIELD-ID.&Type({IOSet}{@fieldType}) OPTIONAL
}
FieldTypes FIELD-ID ::= {
{ Prime-p IDENTIFIED BY prime-field } |
{ Characteristic-two IDENTIFIED BY characteristic-two-field }|
{ Odd-characteristic IDENTIFIED BY id-ft-odd-characteristic },
... -- expect additional field types
}
-- prime fieds
Prime-p ::= INTEGER
-- characteristic two fields
CHARACTERISTIC-TWO ::= TYPE-IDENTIFIER
-- when basis is gnBasis then the basis shall be an optimal
-- normal basis of Type T where T is determined as follows:
-- if an ONB of Type 2 exists for the given value of m then
-- T shall be 2, otherwise if an ONB of Type 1 exists for the
-- given value of m then T shall be 1, otherwise T shall be
```

```
-- the least value for which an ONB of Type T exists for the
-- given value of m
-- when basis is gnBasis then m shall not be divisible by 8
-- note: the above rule is from ANSI X9.62
-- note: for the given m and T the ONB is unique
Characteristic-two ::= SEQUENCE {
m INTEGER,-- extension degree
basis CHARACTERISTIC-TWO.&id({BasisTypes}),
parameters CHARACTERISTIC-TWO.&Type({BasisTypes}{@basis})
}
BasisTypes CHARACTERISTIC-TWO ::= {
{ NULL IDENTIFIED BY gnBasis } |
{ Trinomial IDENTIFIED BY tpBasis } |
{ Pentanomial IDENTIFIED BY ppBasis } |
{ CharTwoPolynomial IDENTIFIED BY charTwoPolynomialBasis },
... -- expect additional basis types
}
Trinomial ::= INTEGER
Pentanomial ::= SEQUENCE {
k1 INTEGER,
k2 INTEGER,
k3 INTEGER
}
-- characteric two general irreducible polynomial representation
-- the irreducible polymial
-- a(n)*x^n + a(n-1)*x^(n-1) + ... + a(1)*x + a(0)
-- is encoded in the bit string with a(n) in the first bit, the
-- following coefficients in the following bit positions and a(0)
-- in the last bit of the bit string (one could omit a(n) and a(0)
-- but it may be simpler and less error-prone to leave them in
-- the encoding)
-- the degree of the polynomial is to be inferred from the length
-- of the bit string
CharTwoPolynomial ::= BIT STRING
-- odd characteristic extension fields
ODD-CHARACTERISTIC ::= TYPE-IDENTIFIER
Odd-characteristic ::= SEQUENCE {
characteristic INTEGER(3..MAX),
degree INTEGER(2..MAX),
basis ODD-CHARACTERISTIC.&id({OddCharBasisTypes}),
parameters ODD-CHARACTERISTIC.&Type({OddCharBasisTypes}{@basis})
}
OddCharBasisTypes ODD-CHARACTERISTIC ::= {
{ OddCharPolynomial IDENTIFIED BY oddCharPolynomialBasis },
... -- expect additional basis types
}
-- the monic irreducible polynomial is encoded as follows
-- the leading coefficient is ignored
-- the remaining coefficients define an element of the finite field
```

```
-- which is encoded in an octet string using FE2OSP
OddCharPolynomial ::= FieldElement
EllipticCurveSubgroupParameters ::= SEQUENCE {
version INTEGER { ecpVer1(1) } (ecpVer1),
fieldID FieldID {{ FieldTypes }},
curve Curve,
generator ECPoint,   -- Base point G
subgroupOrder INTEGER,  -- Order mu of the base point
subgroupIndex INTEGER,  -- The integer nu = #E(F)/mu
...
}
Curve ::= SEQUENCE {
aCoeff FieldElement,
bCoeff FieldElement,
seed BIT STRING OPTIONAL
}
--#####################################################################
-- auxiliary definitions
FieldElement ::= OCTET STRING -- obtained through FE2OSP
ECPoint ::= OCTET STRING -- obtained through EC2OSP
KeyLength ::= INTEGER (1..MAX)
MacAlgorithm ::= AlgorithmIdentifier {{ MACAlgorithms }}
MACAlgorithms ALGORITHM ::= {
{ OID hMAC-SHA1 PARMS NULL } ,
... -- Expect additional algorithms --
}
HashFunction ::= AlgorithmIdentifier {{ HashFunctionAlgorithms }}
HashFunctionAlgorithms ALGORITHM ::= {
  HashFunctionAlgs | -- from 10118-3
 { NULL IDENTIFIED BY id-itu-sha1 } |
 { NULL IDENTIFIED BY id-itu-sha224 } |
 { NULL IDENTIFIED BY id-itu-sha256 } |
 { NULL IDENTIFIED BY id-itu-sha384 } |
 { NULL IDENTIFIED BY id-itu-sha512 },
... -- expect additional algorithms
}
KeyDerivationFunction ::= AlgorithmIdentifier {{ KDFAlgorithms }}
KDFAlgorithms ALGORITHM ::= {
{ OID id-kdf-kdf1 PARMS HashFunction } |
{ OID id-kdf-kdf2 PARMS HashFunction } |
{ OID id-mgf1 PARMS HashFunction },
... -- Expect additional algorithms --
}
SymmetricCipher ::= AlgorithmIdentifier {{ SymmetricAlgorithms }}
SymmetricAlgorithms ALGORITHM ::= {
{ OID id-sc-sc1 PARMS BlockCipher } |
{ OID id-sc-sc2 PARMS BlockCipher } |
{ OID id-camellia128-cbc PARMS BlockCipher } |
{ OID id-aes128-CBC PARMS BlockCipher },
```

```
... -- Expect additional algorithms --
}
BlockCipher ::= AlgorithmIdentifier {{ BlockAlgorithms }}
--#####################################################################
-- external OIDs
-- HMAC-SHA1
hMAC-SHA1 OID ::= {
iso(1) identified-organization(3) dod(6) internet(1) security(5)
mechanisms(5) 8 1 2 }
-- X9.62 finite field and basis types
ansi-X9-62 OID ::= { iso(1) member-body(2) us(840) 10045 }
id-fieldType OID ::= { ansi-X9-62 fieldType(1) }
prime-field OID ::= { id-fieldType 1 }
characteristic-two-field OID ::= { id-fieldType 2 }
-- characteristic two basis
id-characteristic-two-basis OID ::= { characteristic-two-field
basisType(3) }
gnBasis OID ::= { id-characteristic-two-basis 1 }
tpBasis OID ::= { id-characteristic-two-basis 2 }
ppBasis OID ::= { id-characteristic-two-basis 3 }
--#####################################################################
-- Cryptographic algorithm identification --
ALGORITHM ::= CLASS {
&id OBJECT IDENTIFIER UNIQUE,
&Type OPTIONAL
}
WITH SYNTAX { OID &id [PARMS &Type] }
AlgorithmIdentifier { ALGORITHM:IOSet } ::= SEQUENCE {
algorithm ALGORITHM.&id( {IOSet} ),
parameters ALGORITHM.&Type( {IOSet}{@algorithm} ) OPTIONAL
}
END -- EncryptionAlgorithms-2 --
```