PSEC-KEM 仕様書

version 2.2

日本電信電話株式会社
NTT 情報流通プラットフォーム研究所

平成 20 年 4 月 14 日

更新履歴

version 2.2	2008/4/14	3 章のコンバージョン関数と 1 章,3 章,5 章の修正および 誤字脱字の修正.(英語版 version2.2 に対応する修正) 付録および誤字脱字の修正.(英語版 version2.1 に対応する修正) ハッシュ関数に SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 を追加.(英語版 version2.01 に対応する修正) ISO/IEC 18033-2 準拠となるように修正 (英語版 version2.0 に 対応する修正)
version 1.1	2002/5/14	3 章のコンバージョンの修正および誤字脱字の修正.
version 1.0	2001/9/26	

目 次

1 はじめに

本仕様は, 鍵カプセル化メカニズム PSEC-KEM の仕様である. 鍵カプセル化メカニズム (KEM) は, 対称暗号技術で用いる鍵を暗号化するための非対称暗号技術である. このような対象暗号技術をデータカプセル化メカニズム (DEM) と呼ぶ. KEM と DEM を組み合わせることを KEM-DEM フレームワークと呼び, 非対称ハイブリッド暗号を実現することができる. これらの技術の定義および使用法に関しては ISO/IEC 18033-2 [1] の 8 章を参照のこと.

本仕様に基づいた PSEC-KEM の実装は ISO/IEC 18033-2 [1] で規定される PSEC-KEM に準拠している.

2 記法

2.1 記法

ビット	0 または 1 のシンボル
ビット列	ビットが並べられた列
オクテット	長さ 8 のビット列
オクテット列	オクテットが並べられた列
$\mathbb{R}$	実数の集合
$\mathbb{Z}$	整数の集合
$\mathbb{N}$	正の整数の集合
$a := b$	b の値を a に代入する. または， a は b で定義される.
$\mathbb{F}_{q^m}$	要素数 q^m の有限体. q は素数.
$\mathcal{O}$	楕円曲線上の無限遠点
$\|$	ビット列またはオクテット列の連結演算子．この記号は省略されることもある.
$\oplus$	ビット毎の排他的論理和
$\lceil y \rceil$	$y \in \mathbb{R}$ のとき， y 以上の最小の整数.
$\lfloor y \rfloor$	$y \in \mathbb{R}$ のとき，y 以下の最大の整数.
$a \mid b$	a は b を割り切る．つまり， $b = ac$ となる整数 c が存在する.
$a \mod m$	$a \in \mathbb{Z}, m \in \mathbb{N}$ のとき， $m \mid (a - b)$ なる最小の非負整数 b.
$a^{-1} \mod m$	$a \in \mathbb{Z}, m \in \mathbb{N}$ のとき， $ab \mod m = 1$ なる最小の非負整数 b.

2.2 楕円曲線パラメータ

楕円曲線パラメータ E は九つ組 $(q, m, f(\boldsymbol{\beta}), \boldsymbol{a}, \boldsymbol{b}, P, p, pLen, qmLen)$ からなる．ここで各構成要素は以下のものである：

- q, 素数. ただし, $q \neq 3$
- m, 正の整数
- $f(\boldsymbol{\beta})$, 次数 m の $\mathbb{F}_q$ 係数モニック既約多項式, $\mathbb{F}_{q^m}$ の定義多項式
- $\boldsymbol{a}$, $\mathbb{F}_{q^m}$ の元
- $\boldsymbol{b}$, $\mathbb{F}_{q^m}$ の元

- P, 楕円曲線上の点 $(\boldsymbol{x}, \boldsymbol{y})$
  - $\boldsymbol{x}$, $\mathbb{F}_{q^m}$ の元
  - $\boldsymbol{y}$, $\mathbb{F}_{q^m}$ の元

  ただし、$\boldsymbol{x}, \boldsymbol{y}$ は次の式を満たす.
  $q > 3$ のとき： $\boldsymbol{y}^2 = \boldsymbol{x}^3 + \boldsymbol{a}\boldsymbol{x} + \boldsymbol{b}$, ここで $4\boldsymbol{a}^3 + 27\boldsymbol{b}^2 \neq 0$.
  $q = 2$ のとき： $\boldsymbol{y}^2 + \boldsymbol{x}\boldsymbol{y} = \boldsymbol{x}^3 + \boldsymbol{a}\boldsymbol{x}^2 + \boldsymbol{b}$, ここで $\boldsymbol{b} \neq 0$.
- p, P の位数, 素数
- $pLen$, $\lceil \log_{256} p \rceil$ の値
- $qmLen$, $\lceil \log_{256} q^m \rceil$ の値

3 データ型および型変換

本仕様で規定する暗号アルゴリズムは, 非負整数, 有限体, オクテット列, ビット列, 楕円曲線上の点の 5 つのデータ型に関する演算によって構成されており, しばしばこれらのデータ型の間の型変換を必要とする. 本仕様に準拠する異なる実装がデータ交換を確実に行うためには, これらの型変換の仕様を一意に確定しておく必要がある. 本節ではそれらの型変換の規定を行う. 必要なデータ型および型変換の関係を図 1 に示す. 5 つのデータ型はそれぞれ略称, 非負整数 (I), 有限体 (FE), オクテット列 (OS), ビット列 (BS), 楕円曲線上の点 (ECP) を持つ. それぞれの型変換関数の名称は入力および出力の型の略称の組み合わせにより決定される. 型変換関数によっては補助的な入力が必要な場合がある. 各々の型変換関数に入力定義域外のデータが入力された場合は `INVALID` がエラー出力されるとする. 一回の暗号アルゴリズムの処理内で一度でもエラー出力があった場合は, エラーを出力した関数の呼び出し元 または より上位のモジュールにて確実に捕捉され, その暗号アルゴリズムの処理は中断され, 適切な処理が行われる事を意図している.

3.1 整数からビット列への変換 (I2BSP)

本仕様に基づく実装は, 整数からビット列への変換を実行する際, 以下の関数 I2BSP(x, l) と入出力関係が等価なアルゴリズムを使用しなくてはならない. この関数は整数を 2 進数で表現するアルゴリズムに基づいている. 型変換関数 I2BSP(x, l) は形式的に次の様に定義される:

入力: x 非負整数
l 出力のビット長, 非負整数
出力: B 長さ l のビット列
エラー: `INVALID`
処理手順:

1. もし $x \geq 2^l$ ならば, `INVALID` をエラー出力し停止する.
2. もし $l = 0$ ならば, 空のビット列を出力し停止する.

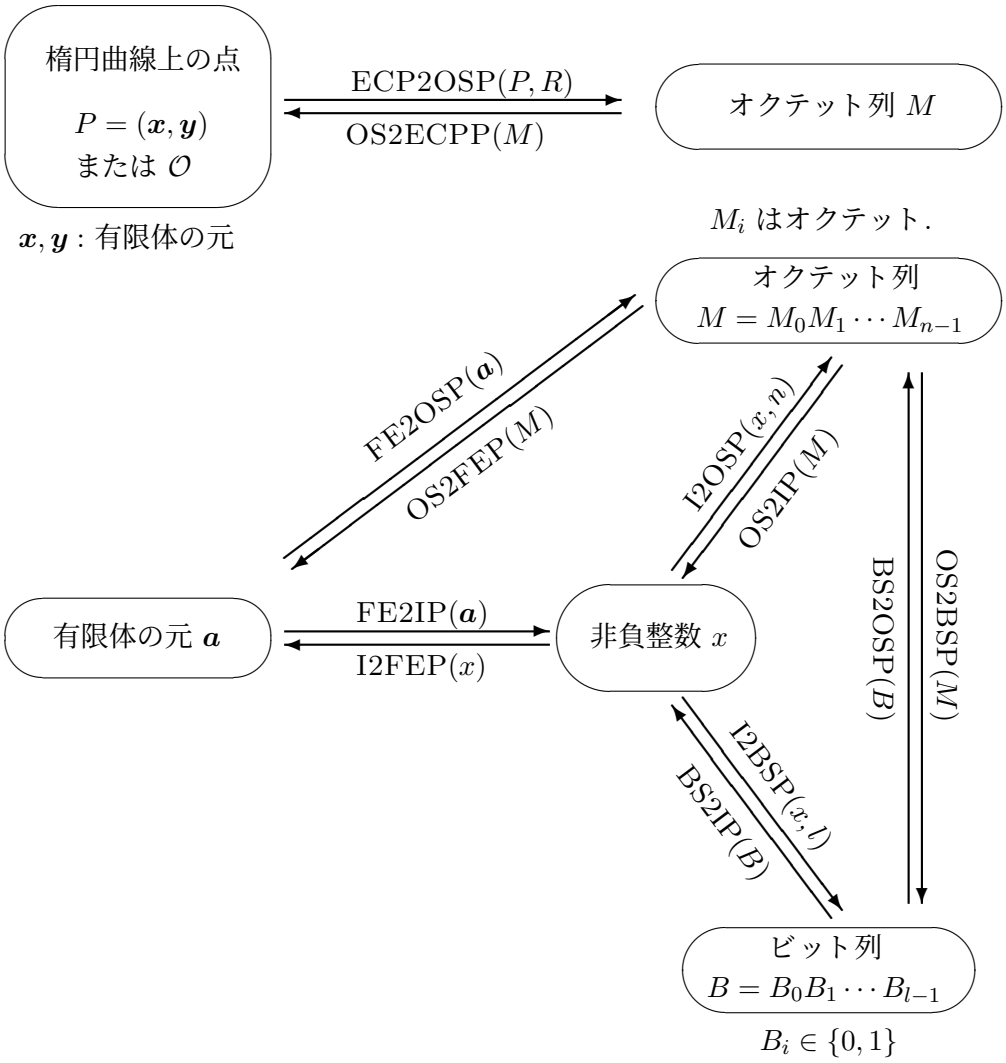

図 1: データ型および型変換の関係

3. 以下を満たす x の 2 進表現 $x_i \in \{0, 1\}$ を決定する.

$$x = x_{l-1}2^{l-1} + x_{l-2}2^{l-2} + \cdots + x_1 2 + x_0.$$

4. $i \in \{0, \ldots, l-1\}$ に関して $B_i := x_{l-1-i}$ とし,

$$B := B_0 B_1 \cdots B_{l-1}.$$

とする.

5. B を出力する.

3.2 ビット列から整数への変換 (BS2IP)

本仕様に基づく実装は, ビット列から整数への変換を実行する際, 以下の関数 BS2IP(B) と入出力関係が等価なアルゴリズムを使用しなくてはならない. この関数はビット列を単に整数の 2 進表現とみなすアルゴリズムに基づいている. 型変換関数 BS2IP(B) は形式的に次の様に定義される:

入力: B 長さ l のビット列
出力: x 非負整数
処理手順:

$B = B_0B_1\cdots B_{l-1}$ なるビット列 B は以下のように整数 x に変換される:

1. もし $l = 0$ ならば, 0 を出力して停止する.
2. 各 B_i を 整数 ($\in \{0,1\}$) とみなし, $i \in \{0,\ldots,l-1\}$ に関して $x_i := B_i$ とし,

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

とする.

3. x を出力する.

3.3 ビット列からオクテット列への変換 (BS2OSP)

本仕様に基づく実装は, ビット列からオクテット列への変換を実行する際, 以下の関数 $\mathrm{BS2OSP}(B)$ と入出力関係が等価なアルゴリズムを使用しなくてはならない. この関数はビット列の長さを 8 の 倍数とするため 左端を 0 で 埋めるアルゴリズムに基づいている. 型変換関数 $\mathrm{BS2OSP}(B)$ は形式的に次の様に定義される:

入力: B 長さ l のビット列
出力: M 長さ $n = \left\lceil \frac{l}{8} \right\rceil$ のオクテット列
処理手順:

ビット列 $B = B_0B_1\cdots B_{l-1}$ をオクテット列 $M = M_0M_1\cdots M_{n-1}$ へ以下のように変換する:

1. もし $l = 0$ ならば, 空のオクテット列を出力して停止する.
2. $j \in \{0,\ldots,8n-1\}$ に関して

$$\widetilde{B}_j = \begin{cases} B_{j-(8n-l)} & \text{if } j \geq 8n-l, \\ 0 & \text{if } j < 8n-l. \end{cases}$$

とする.

3. $i \in \{0,\ldots,n-1\}$, に関して

$$M_i := \widetilde{B}_{8i}\widetilde{B}_{8i+1}\cdots\widetilde{B}_{8i+7}$$

とする.

4. M を出力する.

3.4 オクテット列からビット列への変換 (OS2BSP)

本仕様に基づく実装は, オクテット列からビット列への変換を実行する際, 以下の関数 OS2BSP(M) と入出力関係が等価なアルゴリズムを使用しなくてはならない. この関数はオクテット列を単にビット列とみなすアルゴリズムに基づいている. 型変換関数 OS2BSP(M) は形式的に次の様に定義される:

入力: M 長さ n のオクテット列
出力: B 長さ $l = 8n$ のビット列
処理手順:

オクテット列 $M = M_0 M_1 \cdots M_{n-1}$ をビット列 $B = B_0 B_1 \cdots B_{l-1}$ へ以下のように変換する.

1. もし $l = 0$ ならば, 空のビット列を出力し停止する.
2. $i \in \{0, \ldots, n-1\}, j \in \{0, \ldots, 7\}$ に関して以下を満たす $B_{8i+j} \in \{0, 1\}$ を決定する.

$$M_i = B_{8i} B_{8i+1} \cdots B_{8i+7}$$

3. B を出力する.

3.5 整数からオクテット列への変換 (I2OSP)

本仕様に基づく実装は, 整数からオクテット列への変換を実行する際, 以下の関数 I2OSP(x, n) と入出力関係が等価なアルゴリズムを使用しなくてはならない. この関数は整数をビット列に変換し, 得られたビット列をオクテット列に変換するアルゴリズムに基づいている. 型変換関数 I2OSP(x, n) は形式的に次の様に定義される:

入力: x 非負整数
n 出力のオクテット長.
出力: M 長さ n のオクテット列
エラー: `INVALID`
処理手順:

1. $M :=$ BS2OSP(I2BSP($x, 8n$)) とする.
2. M を出力する.

3.6 オクテット列から整数への変換 (OS2IP)

本仕様に基づく実装は, オクテット列から整数への変換を実行する際, 以下の関数 OS2IP(M) と入出力関係が等価なアルゴリズムを使用しなくてはならない. この関数はオクテット列をビット列に変換し, 得られたビット列を整数に変換するアルゴリズムに基づいている. 型変換関数 OS2IP(M) は形式的に次の様に定義される:

入力: M 長さ n のオクテット列
出力: x 非負整数
処理手順:

1. $x := \mathrm{BS2IP}(\mathrm{OS2BSP}(M))$ とする.

2. x を出力する.

3.7 有限体の元から整数への変換 (FE2IP)

本仕様に基づく実装は, 有限体から整数への変換を実行する際, 以下の関数 $\mathrm{FE2IP}(\boldsymbol{a})$ と入出力関係が等価なアルゴリズムを使用しなくてはならない. 有限体の元は整数係数多項式の係数の列として表現されているとする. q を有限体の標数として, この関数は係数の列を単に整数の q 進数表示とみなすアルゴリズムに基づいている. 型変換関数 $\mathrm{FE2IP}(\boldsymbol{a})$ は形式的に次の様に定義される:

システムパラメータ:	$\mathbb{F}_{q^m}$	位数 q^m の有限体 (q は素数, $m > 0$ は整数)
入力:	$\boldsymbol{a}$	$\mathbb{F}_{q^m}$ の元
出力:	x	0 以上 q^m 未満の整数

処理手順:

有限体の元 $\boldsymbol{a}$ を整数 x に以下のように変換する:

$m = 1$ の場合:
有限体の元 $\boldsymbol{a}$ は整数 ($\in \{0, \ldots, q-1\}$) として表現されている.

1. $x := \boldsymbol{a}$ とする.
2. x を出力する.

$m > 1$ の場合:
有限体の元 $\boldsymbol{a}$ は $\{0, \ldots, q-1\}$ に係数をもつ高々 $(m-1)$ 次の多項式として表現されている. β をこの多項式の不定元とする.

1. $i \in \{0, \ldots, m-1\}$ に関して以下を満たす係数 $a_i \in \{0, \ldots, q-1\}$ を決定する.
$$\boldsymbol{a} = \sum_{i=0}^{m-1} a_i \beta^i.$$
2. 以下を計算する.
$$x := \sum_{i=0}^{m-1} a_i q^i.$$
3: x を出力する.

3.8 整数から有限体の元への変換 (I2FEP)

本仕様に基づく実装は, 整数から有限体への変換を実行する際, 以下の関数 $\mathrm{I2FEP}(x)$ と入出力関係が等価なアルゴリズムを使用しなくてはならない. 有限体の元は整数係数多項式の係数の列として表現されるとする. q を有限体の標数として, この関数は整数を q 進数表示し, 各桁を多項式の係数とみなすアルゴリズムに基づいている. 型変換関数 $\mathrm{I2FEP}(x)$ は形式的に次の様に定義される:

システムパラメータ: $\mathbb{F}_{q^m}$ 位数 q^m の有限体 (q は素数, $m > 0$ は整数)
入力: x 0 以上 q^m 未満の整数
出力: $\boldsymbol{a}$ $\mathbb{F}_{q^m}$ の元
処理手順:

整数 x を有限体の元 $\boldsymbol{a}$ へ以下のように変換する:

$m = 1$ の場合:
$\mathbb{F}_{q^m}$ の元は必ず整数 ($\in \{0, \ldots, q-1\}$) として表現される.

1. $\boldsymbol{a} := x$ とする.
2. $\boldsymbol{a}$ を出力する.

$m > 1$ の場合:
$\mathbb{F}_{q^m}$ の元は必ず $\{0, \ldots, q-1\}$ に係数をもつ高々 $(m-1)$ 次の多項式として表現される. β をこの多項式の不定元とする.

1. x を $i \in \{0, \ldots, m-1\}$ に関して以下を満たす q 進表現 $x_i \in \{0, \ldots, q-1\}$ に展開する.
$$x = \sum_{i=0}^{m-1} x_i q^i.$$
2. 以下を計算する.
$$\boldsymbol{a} := \sum_{i=0}^{m-1} x_i \beta^i.$$
3: $\boldsymbol{a}$ を出力する.

3.9 有限体の要素からオクテット列への変換 (FE2OSP)

型変換関数 FE2OSP($\boldsymbol{a}$) は以下のように定義される:

システムパラメータ: $\mathbb{F}_{q^m}$ 位数 q^m の有限体 (q は素数, $m > 0$ は整数)
n $\left\lceil \dfrac{m \log_2 q}{8} \right\rceil$ と等しい整数
入力: $\boldsymbol{a}$ $\mathbb{F}_{q^m}$ の元
出力: M オクテット列
処理手順:

1.
$$M := \mathrm{I2OSP}(\mathrm{FE2IP}(\boldsymbol{a}), n).$$
とする.
2. M を出力する.

注: システムパラメータは FE2IP と同じ.

3.10 オクテット列から有限体の要素への変換 (OS2FEP)

型変換関数 OS2FEP(M) は以下のように定義される:

システムパラメータ: $\mathbb{F}_{q^m}$ 位数 q^m の有限体 (q は素数, $m > 0$ は整数)
n $\left\lceil \frac{m \log_2 q}{8} \right\rceil$ と等しい整数
入力: M オクテット列
出力: $\boldsymbol{a}$ $\mathbb{F}_{q^m}$ の元
エラー: INVALID
処理手順:

1. $$\boldsymbol{a} := \mathrm{I2FEP}(\mathrm{OS2IP}(M)).$$
とする.

2. $\boldsymbol{a}$ を出力する.

注: システムパラメータは I2FEP と同じ.

3.11 楕円曲線上の点からオクテット列への変換 (ECP2OSP)

楕円曲線上の点からオクテット列への変換方法はおよそ次の通りである.

もし圧縮フォーマットを用いるなら圧縮を利用する旨の指示と共に 圧縮された y 座標がオクテット列の先頭オクテットに記録され, その後に x 座標が配置される.

圧縮フォーマットを用いないなら圧縮を利用しない旨の指示をオクテット列の先頭オクテットに配置し, その後に x 座標を配置し, その後に y 座標を配置する.

型変換関数 ECP2OSP(P, R) の詳細は以下の通りである:

システムパラメータ: E 楕円曲線パラメータ
入力: P $\mathbb{F}_{q^m}$ 上の楕円曲線の点
R Compressed または Uncompressed または Hybrid
出力: M 長さ n のオクテット列

ただし
$$\begin{cases} n = 1 & (P = \mathcal{O} \text{ のとき}) \\ n = \left\lceil \frac{m \log_2 q}{8} \right\rceil + 1 & (P \neq \mathcal{O} \text{ かつ } R \text{ が Compressed のとき}), \\ n = 2\left\lceil \frac{m \log_2 q}{8} \right\rceil + 1 & (P \neq \mathcal{O} \text{ かつ } R \text{ が Uncompressed または Hybrid のとき}) \end{cases}$$

処理手順:

P をオクテット列 $M = M_0 M_1 \cdots M_{n-1}$ に以下のように変換する:

1. $P = \mathcal{O}$ ならば $M := \mathrm{I2OSP}(0, 1)$ を出力する.

2. $P = (\boldsymbol{x}, \boldsymbol{y}) \neq \mathcal{O}$ であり, かつ $R =$ Compressed ならば, 次を行う.

   2.1. オクテット列 X を $X := \mathrm{FE2OSP}(\boldsymbol{x})$ とする.

   2.2. $\boldsymbol{y}$ より 1 ビットの $\tilde{y}$ を以下のように求める (このとき y 座標は 1 ビットに収まる):

2.2.1. q が奇素数の場合, $\boldsymbol{y} = \mathbf{0}$ ならば $\tilde{y} := 0$ とし, $\boldsymbol{y} \neq \mathbf{0}$ ならば $\tilde{y} := y_i \bmod 2$ とする. ただし, $\boldsymbol{y} = y_{m-1}\beta^{m-1} + \cdots + y_1\beta + y_0$ としたときに i は $y_i \neq 0$ を満たすもっとも小さな整数である.

2.2.2. $q = 2$ の場合, $\boldsymbol{x} = \mathbf{0}$ ならば $\tilde{y} := 0$ とし, そうでなければ $\boldsymbol{z} = \boldsymbol{y}\boldsymbol{x}^{-1}$ を満たす $\boldsymbol{z} = z_{m-1}\beta^{m-1} + \cdots + z_1\beta + z_0$ を計算し, $\tilde{y} := z_0$ とする.

2.3. $\tilde{y} = 0$ ならば I2OSP$(2, 1)$ を L に割り当てる. $\tilde{y} = 1$ ならば I2OSP$(3, 1)$ を L に割り当てる.

2.4. $M := L \,\|\, X$ を出力する.

3. $P = (\boldsymbol{x}, \boldsymbol{y}) \neq \mathcal{O}$ かつ $R =$ Uncompressed ならば, 次を行う.

3.1. オクテット列 X を $X :=$ FE2OSP$(\boldsymbol{x})$ とする.

3.2. オクテット列 Y を $Y :=$ FE2OSP$(\boldsymbol{y})$ とする.

3.3. $M :=$ I2OSP$(4, 1) \,\|\, X \,\|\, Y$ を出力する.

4. $P = (\boldsymbol{x}, \boldsymbol{y}) \neq \mathcal{O}$ かつ $R =$ Hybrid ならば, 次を行う.

4.1. オクテット列 X を $X :=$ FE2OSP$(\boldsymbol{x})$ とする.

4.2. オクテット列 Y を $Y :=$ FE2OSP$(\boldsymbol{y})$ とする.

4.3. $\boldsymbol{y}$ より 1 ビットの $\tilde{y}$ を以下のように求める (このとき y 座標は 1 ビットに収まる):

4.3.1. q が奇素数の場合, $\boldsymbol{y} = \mathbf{0}$ ならば $\tilde{y} := 0$ とし, $\boldsymbol{y} \neq \mathbf{0}$ ならば $\tilde{y} := y_i \bmod 2$ とする. ただし, $\boldsymbol{y} = y_{m-1}\beta^{m-1} + \cdots + y_1\beta + y_0$ としたときに i は $y_i \neq 0$ を満たすもっとも小さな整数である.

4.3.2. $q = 2$ の場合, $\boldsymbol{x} = \mathbf{0}$ ならば $\tilde{y} := 0$ とし, そうでなければ $\boldsymbol{z} = \boldsymbol{y}\boldsymbol{x}^{-1}$ を満たす $\boldsymbol{z} = z_{m-1}\beta^{m-1} + \cdots + z_1\beta + z_0$ を計算し, $\tilde{y} := z_0$ とする.

4.4. $\tilde{y} = 0$ ならば I2OSP$(6, 1)$ を L に割り当てる. $\tilde{y} = 1$ ならば I2OSP$(7, 1)$ を L に割り当てる.

4.5. $M := L \,\|\, X \,\|\, Y$ を出力する.

3.12 オクテット列から楕円曲線上の点への変換 (OS2ECPP)

楕円曲線上の点からオクテット列への変換方法はおよそ次の通りである.

もし圧縮フォーマットが用いられているならオクテット列の先頭に配置されるオクテットより圧縮された y 座標を取り出し, その後に続くオクテット列より x 座標を取り出す. しかる後に 完全な y 座標を取り出す.

圧縮フォーマットが用いられていないなら先頭のオクテットを削除して続くオクテット列の前半より x 座標を, 後半より y 座標を復元する.

型変換関数 OS2ECPP(M) の詳細は以下の通りである:

システムパラメータ: E　楕円曲線パラメータ
入力: M　以下のいずれかのオクテット列
I2OSP$(0,1)$ または
長さ $n = \left\lceil \frac{m \log_2 q}{8} \right\rceil + 1$ のオクテット列または
長さ $n = 2\left\lceil \frac{m \log_2 q}{8} \right\rceil + 1$ のオクテット列
出力: P　楕円曲線上の点
エラー: `INVALID`
処理手順:

M を楕円曲線上の点 P に以下のように変換する:

1. $M = \mathrm{I2OSP}(0,1)$, ならば $P := \mathcal{O}$ を出力する.
2. M が長さ $\left\lceil \frac{m \log_2 q}{8} \right\rceil + 1$ のオクテット列の場合, 次を行う:
   2.1. M を $M = L \,\|\, X$ となるように, 長さ 1 のオクテット L と, 続く長さ $\left\lceil \frac{m \log_2 q}{8} \right\rceil$ のオクテット列 X に分解する.
   2.2. $\boldsymbol{x} := \mathrm{OS2FEP}(X)$ とする.
   2.3. $L = \mathrm{I2OSP}(2,1)$ なら, $\tilde{y} := 0$ とし, $L = \mathrm{I2OSP}(3,1)$ なら, $\tilde{y} := 1$ とし, それ以外ならば `INVALID` をエラー出力し停止する.
   2.4. $\boldsymbol{x}$ と $\tilde{y}$ より楕円曲線上の点 $P := (\boldsymbol{x}, \boldsymbol{y})$ を以下のように求める:
      2.4.1. q が奇素数の場合, $\mathbb{F}_{q^m}$ 上の元 $\boldsymbol{w} = \boldsymbol{x}^3 + \boldsymbol{a}\boldsymbol{x} + \boldsymbol{b}$ を計算し, $\boldsymbol{w}$ の $\mathbb{F}_{q^m}$ 上の平方根 $\boldsymbol{\gamma}$ を求める. 平方根が存在しないなら `INVALID` をエラー出力し停止する. もし $\boldsymbol{\gamma} = \boldsymbol{0}$ ならば, $\boldsymbol{y} = \boldsymbol{0}$ とする. もし $\gamma_i \equiv \tilde{y} \mod 2$ ならば $\boldsymbol{y} = \boldsymbol{\gamma}$ とし, $\gamma_i \not\equiv \tilde{y} \mod 2$ ならば $\boldsymbol{y} = -\boldsymbol{\gamma}$ とする. ただし, $\boldsymbol{\gamma} = \gamma_{m-1}\beta^{m-1} + \cdots + \gamma_1\beta + \gamma_0$ としたときに i は $\gamma_i \neq 0$ を満たすもっとも小さな整数である.
      2.4.2. $q = 2$ で $\boldsymbol{x} = \boldsymbol{0}$ なら, $\boldsymbol{y} := \boldsymbol{b}^{2^{m-1}} \in \mathbb{F}_{q^m}$ とする.
      2.4.3. $q = 2$ で $\boldsymbol{x} \neq \boldsymbol{0}$ なら, $\mathbb{F}_{q^m}$ 上の元 $\boldsymbol{\gamma} := \boldsymbol{x} + \boldsymbol{a} + \boldsymbol{b}\boldsymbol{x}^{-2}$ を計算し, $\boldsymbol{z}^2 + \boldsymbol{z} = \boldsymbol{\gamma}$ なる $\mathbb{F}_{q^m}$ の元 $\boldsymbol{z} = z_{m-1}\beta^{m-1} + \cdots + z_1\beta + z_0$ を求める. もし $\boldsymbol{z}$ が存在しないなら `INVALID` をエラー出力し停止する. $z_0 = \tilde{y}$ ならば $\boldsymbol{y} = \boldsymbol{x}\boldsymbol{z} \in \mathbb{F}_{q^m}$ とする. $z_0 \neq \tilde{y}$ ならば $\boldsymbol{y} = \boldsymbol{x}(\boldsymbol{z} + \boldsymbol{1}) \in \mathbb{F}_{q^m}$ とする.
   2.5. $P := (\boldsymbol{x}, \boldsymbol{y})$ を出力する.
3. M が長さ $2\left\lceil \frac{m \log_2 q}{8} \right\rceil + 1$ のオクテット列の場合, 次を行う:
   3.1. M を $M = L \,\|\, X \,\|\, Y$ となるように, 長さ 1 のオクテット L と, 続く長さ $\left\lceil \frac{m \log_2 q}{8} \right\rceil$ のオクテット列の X と, 続く長さ $\left\lceil \frac{m \log_2 q}{8} \right\rceil$ のオクテット列の Y に分解する.
   3.2. $L = \mathrm{I2OSP}(4,1)$ または $\mathrm{I2OSP}(6,1)$ または $\mathrm{I2OSP}(7,1)$ であるかどうかを検査する. それ以外ならば, `INVALID` をエラー出力し停止する.

3.3. $\boldsymbol{x} := \mathrm{OS2FEP}(X)$ とする.

3.4. $\boldsymbol{y} := \mathrm{OS2FEP}(Y)$ とする.

3.5. 点 $P := (\boldsymbol{x}, \boldsymbol{y})$ が E で定義される楕円曲線の定義方程式を満たさなければ, `INVALID` をエラー出力し停止する.

3.6. $P := (\boldsymbol{x}, \boldsymbol{y})$ を出力する.

3.13 楕円曲線上の点の部分情報からオクテット列への変換 (PECP2OSP)

型変換関数 $\mathrm{PECP2OSP}(P)$ は以下のように定義される:

システムパラメータ: E 楕円曲線パラメータ
入力: P $\mathbb{F}_{q^m}$ 上の楕円曲線の点
出力: M 長さ $n = \left\lceil \dfrac{m \log_2 q}{8} \right\rceil$ のオクテット列.

処理手順:

楕円曲線上の点 P からオクテット列 $M = M_0 M_1 \cdots M_{n-1}$ への変換方法は次の通りである.

1. $P = \mathcal{O}$ ならば, $M := \mathrm{I2OSP}(0, n)$ を出力する.
2. $P = (\boldsymbol{x}, \boldsymbol{y}) \neq \mathcal{O}$ ならば, $M := \mathrm{FE2OSP}(\boldsymbol{x})$ を出力する.

4 システムパラメータ・秘密鍵・公開鍵

4.1 PSEC-KEM システムパラメータ

PSEC-KEM システムパラメータは以下の値からなる.

- E, 2.2 節に記述されている楕円曲線パラメータ
- KDF, 6.2 節に記述されている鍵導出関数
- $hLen$, 正の整数
- $keyLen$, 正の整数

4.2 PSEC-KEM 秘密鍵

PSEC-KEM 秘密鍵は以下の値からなる.

- s, 非負整数 $0 \leq s < p$

4.3 PSEC-KEM 公開鍵

PSEC-KEM 公開鍵は以下の値からなる.

- W, 楕円曲線 E 上の点

5 鍵カプセル化メカニズム PSEC-KEM

本章では、以下の 3 つのアルゴリズムからなる PSEC-KEM について述べる.

- 鍵生成アルゴリズム KGP-PSEC
  入力はなし, 出力は公開鍵と秘密鍵 (W, s).
- 暗号化処理 ES-PSEC-KEM-Encrypt
  入力は公開鍵 W とフォーマット R, 出力は鍵と暗号文の対 (k, c_0).
- 復号処理 ES-PSEC-KEM-Decrypt
  入力は秘密鍵 s と暗号文 c_0, 出力は鍵 k.

5.1 鍵生成処理 (KGP-PSEC)

KGP-PSEC() は以下のように定義される.

入力: KGP-PSEC は入力を取らない
出力: W PSEC-KEM 公開鍵, 楕円曲線 E 上の点
s PSEC-KEM 秘密鍵, 非負整数 $0 \leq s < p$

処理手順:

1. ランダムな整数 $s \in \{0, \cdots, p-1\}$ を生成する.
2. $W := sP$ とする.
3. W と s を出力する.

5.2 鍵カプセル化メカニズム (ES-PSEC-KEM)

5.2.1 暗号化処理

ES-PSEC-KEM-Encrypt(W, R) は以下のように定義される.

入力: W PSEC-KEM 公開鍵, 楕円曲線 E 上の点
R Compressed または Uncompressed または Hybrid
出力: k オクテット列
c_0 オクテット列

処理手順:

1. ランダムな長さ $hLen$ のオクテット列 r を生成する.
2. $H := KDF(\mathrm{I2OSP}(0, 4) \,\|\, r, pLen + 16 + keyLen)$ とする.
3. $H = t \,\|\, k$ を, 長さ $pLen + 16$ のオクテット列 t と, 長さ $keyLen$ のオクテット列 k に分解する.
4. $\alpha := \mathrm{OS2IP}(t) \bmod p$ とする.
5. $C_1 := \alpha P$ とする.

6. $Q := \alpha W$ とする.

7. $c_2 := r \oplus \mathit{KDF}(\mathrm{I2OSP}(1,4) \,\|\, \mathrm{ECP2OSP}(C_1, R) \,\|\, \mathrm{PECP2OSP}(Q), \mathit{hLen})$ とする.

8. $c_0 := \mathrm{ECP2OSP}(C_1, R) \,\|\, c_2$ とする.

9. (k, c_0) を出力する.

5.2.2 復号処理

ES-PSEC-KEM-DECRYPT(s, c_0) は以下のように定義される.

入力: s PSEC-KEM 秘密鍵, 非負整数 $0 \leq s < p$
c_0 オクテット列
出力: k オクテット列
エラー: `INVALID`
処理手順:

1. c_0 のオクテット長が hLen 未満ならば, `INVALID` をエラー出力し停止する.

2. $c_0 = g \,\|\, c_2$ を, オクテット列 g と, 長さ hLen のオクテット列 c_2 に分解する.

3. $C_1 := \mathrm{OS2ECPP}(g)$ とする. OS2ECPP が `INVALID` をエラー出力したら, `INVALID` をエラー出力し停止する.

4. $Q := sC_1$ とする.

5. $r := c_2 \oplus \mathit{KDF}(\mathrm{I2OSP}(1,4) \,\|\, g \,\|\, \mathrm{PECP2OSP}(Q), \mathit{hLen})$ とする.

6. $h := \mathit{KDF}(\mathrm{I2OSP}(0,4) \,\|\, r, \mathit{pLen} + 16 + \mathit{keyLen})$ とする.

7. $h = t \,\|\, k$ を, 長さ $\mathit{pLen} + 16$ のオクテット列 t と, 長さ keyLen のオクテット列 k に分解する.

8. $\alpha := \mathrm{OS2IP}(t) \bmod p$ とする.

9. $C_1 = \alpha P$ をチェックする. 成立するときは, k を出力しする. そうでなければ, `INVALID` をエラー出力し停止する.

6 補助関数

6.1 ハッシュ関数

許容されるハッシュ関数は, ISO/IEC 10118-3 [2] に記載されている SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 である.

6.2 鍵導出関数

鍵導出関数として 6.2.1 節で定義される MGF1 を推奨する. MGF1 は ISO/IEC 18033-2[1] における KDF1 と同じものである.

6.2.1 MGF1

MGF1 は 6.1 節で示されるハッシュ関数を利用したマスク生成関数である.
MGF1(M, n) は次のように定義される.

システムパラメータ:	*Hash*	ハッシュ関数
	hashLen	ハッシュ関数の出力のオクテット長, 正の整数
入力:	M	マスクを生成するための乱数の種, オクテット列
	n	出力のオクテット長, 正の整数
出力:	*mask*	マスク, 長さ n のオクテット列
エラー:	`INVALID`	

処理手順:

1. n_0 を M のオクテット長とする. もし $n_0 + 4$ が ハッシュ関数の入力ビット幅を越えていたならば, `INVALID` をエラー出力して処理を終了する.
2. $cThreshold := \left\lceil \dfrac{n}{hashLen} \right\rceil$ とする.
3. もし $cThreshold > 2^{32}$ ならば, `INVALID` をエラー出力して処理を終了する.
4. M' を空オクテット列とする.
5. 整数 $counter := 0$ とする.
   (a) *counter* を長さ 4 のオクテット列に変換する:
   $$C := \mathrm{I2OSP}(counter, 4).$$
   (b) M と C を連結し, ハッシュ関数 $Hash$ を作用させ, 長さ *hashLen* のハッシュ値 H を生成する:
   $$H := Hash(M \,\|\, C).$$
   (c) M' と H を連結し, その結果を再び M' とする:
   $$M' := M' \,\|\, H.$$
   (d) $counter := counter + 1$ とする. もし $counter < cThreshold$ ならば, 5 (a) に戻る.
6. オクテット列 M' の先頭 n オクテットを *mask* とする
   $$mask := M'_0 M'_1 \cdots M'_{n-1}.$$
7. *mask* を出力する.

参考文献

[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.

付録

A　パラメータ設定基準

PSEC-KEM パラメータの設定基準は以下の通りである.

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

B　パラメータ推奨値

PSEC-KEM パラメータの推奨値は以下の通りである.

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

ある特定の楕円曲線パラメータについては脆弱性が指摘されているが, 適切な楕円曲線暗号のパラメータと実装については SECG のドキュメントを参照のこと. (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 --
```