PSEC-KEM 仕様書

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

2002 年 5 月 14 日

目 次

1 はじめに

本ドキュメントは, PSEC-KEM 鍵カプセル化メカニズムの実装に関する情報をまとめたものである. PSEC-KEM 鍵カプセル化メカニズムは, 鍵共有のために用いることができる. 本ドキュメントには, 以下の項目が記されている.

- プリミティブ (cryptographic primitives): KGP-PSEC, EP-PSEC, DP-PSEC
- 鍵カプセル化メカニズム (key encapsulation mechanisms): ES-PSEC-KEM

1.1 概要

本ドキュメントの構成は以下のとおり:

- 第 1 章は, イントロダクションである.
- 第 2 章では, 記法を説明する.
- 第 3 章では, データ型と変換法を定義する.
- 第 4 章では, KGP-PSEC, EP-PSEC, DP-PSEC で用いられる PSEC 秘密鍵, PSEC 公開鍵を定義する.
- 第 5 章では, PSEC の基本演算を行なうプリミティブを定義する.
- 第 6 章では, PSEC-KEM 鍵カプセル化メカニズムを定義する.
- 第 7 章では, PSEC-KEM 鍵カプセル化メカニズムで用いられるエンコーディングメソッドを定義する.
- 第 8 章では, 補助的に用いられる関数などを説明する.

2 記法

N	自然数の集合
$a := b$	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
$\{0,1\}^i$	長さ i のビット列の集合
$\{0,1\}^*$	$\bigcup_{i=0}^{\infty}\{0,1\}^i$
$\{0,1,\cdots,255\}^i$	長さ i のオクテット列の集合
$\{0,1,\cdots,255\}^*$	$\bigcup_{i=0}^{\infty}\{0,1,\cdots,255\}^i$
$\|\|$	ビット列またはオクテット列の連結演算子 例えば, ビット列に対して $(0,1,0,0) \,\|\|\, (1,1,0) = (0,1,0,0,1,1,0)$, オクテット列に対して $(4,3) \,\|\|\, (6,2) = (4,3,6,2)$ となる.
$\oplus$	ビット毎の排他的論理和
$\lceil y \rceil$	y 以上の最小の整数
$\lfloor y \rfloor$	y 以下の最大の整数
$a \bmod m$	$a, m \in N$ のとき, $m \mid (b-a)$ なる最小の自然数 b

連結演算子 '||' は, しばしば省略される.

3 データ型 及び 変換 (Data types and conversions)

本ドキュメントで規定するスキームには, さまざまなデータ型の演算が使用される. 必要な変換と対応する変換の通称を 図 1 に記載する.

3.1 ビット列からオクテット列への変換 (BS2OSP)

本章に記述するビット列からオクテット列への変換方法は基本的には ビット列 の長さが 8 の倍数となるまでビット 0 をビット列の先頭に付与し, しかる後に ビット列を 8 ビットずつ切り出すという操作に基づいている. プリミティブ BS2OSP(B, l) の詳細は以下の通りである.

入力:

B : l ビット長のビット列
l : 整数

出力:

M : $n = \lceil l/8 \rceil$ オクテット長のオクテット列

処理手順:

ビット列 $B = B_0 B_1 \ldots B_{l-1}$ をオクテット列 $M = M_0 M_1 \ldots M_{n-1}$ へ以下の様に変換する:

1. $0 < i \leq n-1$ の 整数 i に関して以下とする:

$$M_i := B_{l-8-8(n-1-i)} B_{l-7-8(n-1-i)} \cdots B_{l-1-8(n-1-i)}.$$

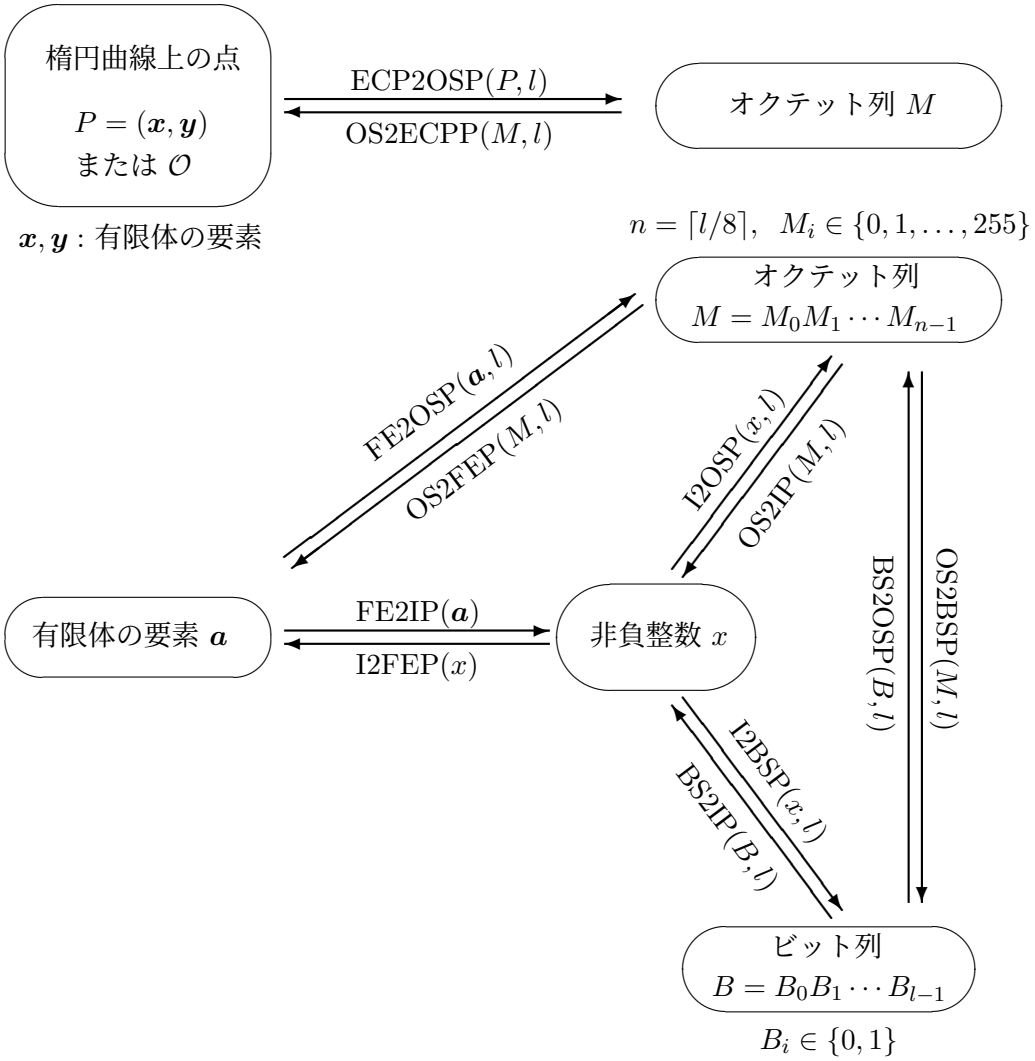

図 1: データ型間の変換

2. M_0 のうち 先頭 $8n - l$ ビットを 0 とし, 末尾 $l + 8 - 8n$ ビットを $B_0 B_1 \ldots B_{l+7-8n}$ とする.
3. M を出力する.

3.2 オクテット列からビット列への変換 (OS2BSP)

本章に記述するオクテット列からビット列への変換方法は基本的にはオクテット列を単に先頭にパディングビットを持つビット列と見做す事に基づいている. プリミティブ OS2BSP(M, l) の詳細は以下の通りである.

入力:

M : $n = \lceil l/8 \rceil$ オクテット長の オクテット列
l : 整数

出力:

B : l ビット長のビット列

処理手順:

オクテット列 $M = M_0M_1 \ldots M_{n-1}$ をビット列 $B = B_0B_1 \ldots B_{l-1}$ へ以下の様に変換する:

1. $0 < i \leq n-1$ の整数 i に関して以下とする:

$$B_{l-8-8(n-1-i)}B_{l-7-8(n-1-i)} \cdots B_{l-1-8(n-1-i)} := M_i.$$

2. M_0 の先頭 $8n-l$ ビットを無視し末尾 $l+8-8n$ ビットの値を $B_0B_1 \ldots B_{l+7-8n}$ に代入する.
3. B を出力する.

3.3 整数からオクテット列への変換 (I2OSP)

本章に記述する整数からオクテット列への変換方法は基本的には整数の 256 進表現に基づいている. プリミティブ I2OSP(x, l) の詳細は以下の通りである.

入力:

x : 非負整数
l : 整数

出力:

M : $n = \lceil l/8 \rceil$ オクテット長のオクテット列

例外:

"invalid" : 入力が不正

処理手順:

1. $x \geq 2^l$ ならば, "invalid" をエラー出力して停止する.
2. $0 \leq i \leq n-1$ の整数 i に関して 次を満たす x の 256 進表現 $x_i \in \{0, \ldots, 255\}$ を求める.

$$x = x_{n-1}2^{8(n-1)} + x_{n-2}2^{8(n-2)} + \cdots + x_1 2^8 + x_0$$

3. $0 \leq i \leq n-1$ の整数 i に関して $M_i := x_{n-1-i}$. として以下とする.

$$M := M_0M_1 \cdots M_{n-1}$$

4. M を出力する.

3.4 オクテット列から整数への変換 (OS2IP)

本章に記述するオクテット列から整数への変換方法は基本的にはオクテット列を単に整数の 256 進表現と見倣す事に基づいている. プリミティブ OS2IP(M, l) の詳細は以下の通りである.

入力:

M : $n = \lceil l/8 \rceil$ オクテット長のオクテット列
l : 整数

出力:

x : 整数

処理手順:

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

1. 各 M_i を 0 以上 255 以下の整数と見做し以下の x を出力する:

$$x := \sum_{i=0}^{n-1} 2^{8(n-1-i)} M_i \mod 2^l.$$

3.5 有限体の要素から整数への変換 (FE2IP)

本章では有限体の要素は, 整係数の多項式で表現されているとする. 多項式は係数の数列によって表現すれば良い. 本章に記述する 有限体から整数への変換方法は, q を有限体の標数として, 基本的には 多項式の係数列を単に 整数の q 進表現と見做す事に基づいている. プリミティブ FE2IP($\boldsymbol{a}$) の詳細は以下の通りである:

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:

$\boldsymbol{a}$: $\mathbb{F}_{q^m}$ の要素

出力:

x : 0 以上 $q^m - 1$ 以下の整数

処理手順:

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

$m = 1$ の場合:

有限体の要素 $\boldsymbol{a}$ は 0 以上 $q^m - 1$ 以下の整数として表現されている.

1: $x := \boldsymbol{a}$ として x を出力する.

$m > 1$ の場合:

有限体の要素 $\boldsymbol{a}$ は $\{0, \dots, q-1\}$ に係数を持つ高々 $m-1$ 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: $i \in \{0, \dots, m-1\}$ の i に関して以下を満たす係数 $a_i \in \{0, \dots, q-1\}$ を決定する.

$$\boldsymbol{a} = \sum_{i=0}^{m-1} a_i \beta^i$$

2: 以下の x を出力する.

$$x := \sum_{i=0}^{m-1} a_i q^i$$

3.6 整数から有限体の要素への変換 (I2FEP)

本章では有限体の要素は, 整係数の多項式で表現されているとする. 多項式は係数の数列によって表現すれば良い. 本章に記述する 整数 から 有限体 への変換方法は, q を有限体の標数として,

基本的には 整数を q 進の位取り記法で表現し, 各桁を 多項式の各係数 と見做す事に基づいている. プリミティブ I2FEP(x) の詳細は以下の通りである:

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 $q^m - 1$ 以下の整数.

出力:

$\boldsymbol{a}$: $\mathbb{F}_{q^m}$ の要素.

処理手順:

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

$m = 1$ の場合:

$\mathbb{F}_{q^m}$ の要素は 0 以上 $q^m - 1$ 以下の整数である.

1: $\boldsymbol{a} := x$ として $\boldsymbol{a}$ を出力する.

$m > 1$ の場合:

$\mathbb{F}_{q^m}$ の要素は $\{0, \ldots, q-1\}$ に係数を持つ高々 $m-1$ 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 以下を満たす q 進表現 $x_i \in \{0, \ldots, q-1\}$, $(i \in \{0, \ldots, m-1\})$ に展開する.

$$x = \sum_{i=0}^{m-1} x_i q^i$$

2: 以下の $\boldsymbol{a}$ を出力する.

$$\boldsymbol{a} := \sum_{i=0}^{m-1} x_i \beta^i$$

3.7 有限体の要素からオクテット列への変換 (FE2OSP)

プリミティブ FE2OSP($\boldsymbol{a}, l$) の詳細は以下の通りである.

入力:

$\boldsymbol{a}$: 有限体の要素

l : 整数

出力:

M : オクテット列

処理手順:

以下の M を出力する.

$$M := \text{I2OSP}(\text{FE2IP}(\boldsymbol{a}), l)$$

3.8 オクテット列から有限体の要素への変換 (OS2FEP)

プリミティブ OS2FEP(M, l) の詳細は以下の通りである.

入力:

M : オクテット列
l : 整数

出力:

$\boldsymbol{a}$: 有限体の要素

処理手順:

以下の $\boldsymbol{a}$ を出力する.

$$\boldsymbol{a} := \mathrm{I2FEP}(\mathrm{OS2IP}(M, l))$$

3.9 楕円曲線上の点からオクテット列への変換 (ECP2OSP)

本章に記述する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.

プリミティブ ECP2OSP(P, l) の詳細は以下の通りである:

準備:

圧縮フォーマットを利用するか否かを決める.

オプション:

E : 楕円曲線パラメータ
R : Compressed, Uncompressed, Hybrid のいずれか

入力:

P : 楕円曲線上の $\mathbb{F}_{q^m}$ 有理点
l : 整数

出力:

M : 以下の n に関する 長さ n のオクテット列.

$$\begin{cases} n = 1 & P = \mathcal{O} \text{ の場合,} \\ n = \lceil l/8 \rceil + 1 & P \neq \mathcal{O} \text{ でかつ } R \text{ が Compressed の場合,} \\ n = 2\lceil l/8 \rceil + 1 & P \neq \mathcal{O} \text{ でかつ } R \text{ が Uncompressed または Hybrid の場合.} \end{cases}$$

処理手順:

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

1. $P = \mathcal{O}$ ならば $M := 00_{16}$ を出力する.
2. $P = (\boldsymbol{x}, \boldsymbol{y}) \neq \mathcal{O}$ でかつ R が Compressed なら, 次を行う:
   2.1. オクテット列 X を $X := \mathrm{FE2OSP}(\boldsymbol{x}, l)$ 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 (此の時 y 座標は 1 ビットに収まる):
      2.2.1. q が 奇数の場合, $\boldsymbol{y} = y_{m-1}\beta^{m-1} + \cdots + y_1\beta + y_0$ に関して $\tilde{y} := y_0 \bmod 2$ とする.
      2.2.2. $q = 2$ の場合, $\boldsymbol{x} = \boldsymbol{0}$ ならば $\tilde{y} := 0$ とし, さもなくば $\boldsymbol{z} = \boldsymbol{y}\boldsymbol{x}^{-1}$ なる $\boldsymbol{z} = z_{m-1}\beta^{m-1} + \cdots + z_1\beta + z_0$ に関して $\tilde{y} := z_0$ とする.
   2.3. 単オクテット L に $\tilde{y} = 0$ なら 02_{16} を, $\tilde{y} = 1$ なら 03_{16} を割り当てる.
   2.4. $M := L \| X$ を出力する.

3. $P = (\boldsymbol{x}, \boldsymbol{y}) \neq \mathcal{O}$ でかつ R が Uncompressed なら, 次を行う:
   3.1. オクテット列 X を $X := \mathrm{FE2OSP}(\boldsymbol{x}, l)$ とする.
   3.2. オクテット列 Y を $Y := \mathrm{FE2OSP}(\boldsymbol{y}, l)$ とする.
   3.3. $M := 04_{16} \| X \| Y$ を出力する.
4. $P = (\boldsymbol{x}, \boldsymbol{y}) \neq \mathcal{O}$ でかつ R が Hybrid なら, 次を行う:
   4.1. オクテット列 X を $X := \mathrm{FE2OSP}(\boldsymbol{x}, l)$ とする.
   4.2. オクテット列 Y を $Y := \mathrm{FE2OSP}(\boldsymbol{y}, l)$ 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 (此の時 y 座標は 1 ビットに収まる):
      4.3.1. q が 奇数なら, $\boldsymbol{y} = y_{m-1}\beta^{m-1} + \cdots + y_1\beta + y_0$ に関して $\tilde{y} := y_0 \bmod 2$ とする.
      4.3.2. $q = 2$ の時, $\boldsymbol{x} = \boldsymbol{0}$ ならは $\tilde{y} := 0$ とし, さもなくば $\boldsymbol{z} = \boldsymbol{y}\boldsymbol{x}^{-1}$ なる $\boldsymbol{z} = z_{m-1}\beta^{m-1} + \cdots + z_1\beta + z_0$ に関して $\tilde{y} := z_0$ とする.
   4.4. 単オクテット L に $\tilde{y} = 0$ なら 06_{16} を, $\tilde{y} = 1$ なら 07_{16} を割り当てる.
   4.5. $M := L \| X \| Y$ を出力する.

3.10 オクテット列から楕円曲線上の点への変換 (OS2ECPP)

本章に記述する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 座標を求める.

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.

プリミティブ OS2ECPP(M, l) の詳細は以下の通りである:

オプション:

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
- 単オクテット 00_{16}
- 長さ $n = \lceil l/8 \rceil + 1$ のオクテット列
- 長さ $n = 2\lceil l/8 \rceil + 1$ のオクテット列

l : 整数

出力:

P : 楕円曲線上の点

例外:

"invalid" : 入力が不正

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. $M = 00_{16}$ の場合, $P := \mathcal{O}$ を出力する.
2. M が $\lceil l/8 \rceil + 1$ オクテット長の場合, 次を行う:
   2.1. $M = L \| X$ を解析し 単オクテット L と, 続く $\lceil l/8 \rceil$ オクテットの X を求める.

2.2. $\boldsymbol{x} := \mathrm{OS2FEP}(X, l)$ とする.
2.3. $L = 02_{16}$ なら $\tilde{y} := 0$ とし, $L = 03_{16}$ なら $\tilde{y} := 1$ とし, それ以外なら "invalid" をエラー出力し停止する.
2.4. $\boldsymbol{x}$ 及び $\tilde{y}$ 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 が奇数の場合, 有限体の要素 $\boldsymbol{w} := \boldsymbol{x}^3 + \boldsymbol{a}\boldsymbol{x} + \boldsymbol{b}$ を計算し, $\boldsymbol{w}$ の平方根 $\boldsymbol{\gamma} \in \mathbb{F}_{q^m}$ を求める. $\mathbb{F}_{q^m}$ 上に平方根が存在しないなら "invalid" をエラー出力し停止する. $\boldsymbol{\gamma} = \gamma_{m-1}\beta^{m-1} + \cdots + \gamma_1\beta + \gamma_0$ として, もし $\gamma_0 \equiv \tilde{y} \bmod 2$ ならば $\boldsymbol{y} := \boldsymbol{\gamma}$ とし, $\gamma_0 \not\equiv \tilde{y} \bmod 2$ ならば $\boldsymbol{y} := -\boldsymbol{\gamma}$ とする.
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 $\boldsymbol{x} \neq 0$ なら有限体の要素 $\boldsymbol{\gamma} := \boldsymbol{x} + \boldsymbol{a} + \boldsymbol{b}\boldsymbol{x}^{-2} \in \mathbb{F}_{q^m}$, を計算し $\boldsymbol{z}^2 + \boldsymbol{z} = \boldsymbol{\gamma}$ なる $\mathbb{F}_{q^m}$ の 要素 $\boldsymbol{z} = z_{m-1}\beta^{m-1} + \cdots + z_1\beta + z_0$ を求める. もし $\mathbb{F}_{q^m}$ 上に $\boldsymbol{z}$ 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 $\boldsymbol{y} := \boldsymbol{x}\boldsymbol{z}$ $z_0 \neq \tilde{y}$ ならば $\boldsymbol{y} := \boldsymbol{x}(\boldsymbol{z} + \boldsymbol{1})$ とする.
2.5. $P = (\boldsymbol{x}, \boldsymbol{y})$ を出力する.

3. M が $2\lceil l/8 \rceil + 1$ オクテット長の場合, 次を行う:
3.1. $M = L \| X \| Y$ を解析し 単オクテット L と, 続く $\lceil l/8 \rceil$ オクテットの X と続く $\lceil l/8 \rceil$ オクテットの Y を求める.
3.2. $L \in \{04_{16}, 06_{16}, 07_{16}\}$ であるか否かを調べ, もし $L \notin \{04_{16}, 06_{16}, 07_{16}\}$ であるなら "invalid" をエラー出力し停止する.
3.3. $\boldsymbol{x} := \mathrm{OS2FEP}(X, l)$ とする.
3.4. $\boldsymbol{y} := \mathrm{OS2FEP}(Y, l)$ 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.

3.11 整数からビット列への変換 (I2BSP)

本章に記述する整数からビット列への変換方法は基本的には整数の 2 進表現に基づいている. プリミティブ $\mathrm{I2BSP}(x, l)$ の詳細は以下の通りである.

入力:

x : 非負整数
l : 整数

出力:

B : l ビット長のビット列

例外:

"invalid" : 入力が不正

処理手順:

1. $x \geq 2^l$ ならば, "invalid" をエラー出力して停止する.
2. $0 \leq i \leq l - 1$ の整数 i に関して 次を満たす x の 2 進表現 $x_i \in \{0, 1\}$ を求める.

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

3. $0 \leq i \leq l-1$ の i について $B_i := x_{l-1-i}$ として以下とする.

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

4. B を出力する.

3.12 ビット列から整数への変換 (BS2IP)

本章に記述するビット列から整数への変換方法は基本的には ビット列を単に 整数の 2 進表現と見做す事に基づいている. プリミティブ BS2IP(B, l) の詳細は以下の通りである.

入力:

B : l ビット長のビット列
l : 整数

出力:

x : 整数

処理手順:

ビット列 $B = B_0 B_1 \ldots B_{l-1}$ を整数 x に以下のように変換:

1. 各 B_i を 整数 0 または 1 と見做し, 以下の x を出力する:

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

4 鍵タイプ (Key types)

本章では, PSEC 秘密鍵と PSEC 公開鍵という 2 種類の鍵を定義する. それらは PSEC-KEM の 3 つのプリミティブ (KGP-PSEC, EP-PSEC, DP-PSEC) で用いられる.

4.1 PSEC 秘密鍵

PSEC 秘密鍵は, 以下の値である.

- s, 非負整数

4.2 PSEC 公開鍵

PSEC 公開鍵は以下の四つ組 $(E, W, KDF, hLen)$ からなる.

- E, 楕円曲線パラメータ
- W, 楕円曲線上の点
- KDF, 鍵導出関数の選択
- $hLen$, 非負整数

楕円曲線パラメータ E は以下の九つ組 $(q, m, f(\boldsymbol{\beta}), \boldsymbol{a}, \boldsymbol{b}, P, p, pLen, qmLen)$ からなる.

- q, 素数
- m, 正の整数
- $f(\boldsymbol{\beta})$, $\mathbb{F}_q$ の次数 m のモニック既約多項式
- $\boldsymbol{a}$, $\mathbb{F}_{q^m}$ の要素
- $\boldsymbol{b}$, $\mathbb{F}_{q^m}$ の要素
- P, 楕円曲線上の点
  - $\boldsymbol{x}$, $\mathbb{F}_{q^m}$ の要素
  - $\boldsymbol{y}$, $\mathbb{F}_{q^m}$ の要素

  $\boldsymbol{y}^2 = \boldsymbol{x}^3 + \boldsymbol{a}\boldsymbol{x} + \boldsymbol{b}$ $(q > 3)$
  $\boldsymbol{y}^2 + \boldsymbol{x}\boldsymbol{y} = \boldsymbol{x}^3 + \boldsymbol{a}\boldsymbol{x}^2 + \boldsymbol{b}$ $(q = 2)$

- p, 素数, P の位数
- $pLen$, $\lceil \log_2 p \rceil$ の値
- $qmLen$, $\lceil \log_2 q^m \rceil$ の値

正しい PSEC 公開鍵は, $W = sP$ を満足する. ここで, s は 4.1 章 における PSEC 秘密鍵である.

注 :
鍵導出関数 KDF は, 8.2 章における鍵導出関数の中の 1 つとする.

5 プリミティブ (Cryptographic primitives)

本章では, 3 種類のプリミティブを定義する.

5.1 KGP-PSEC

KGP-PSEC$(E, KDF, hLen)$ を次のように定義する.

入力:	E	楕円曲線パラメータ
	KDF	鍵導出関数の選択
	$hLen$	非負整数
出力:	PK	PSEC 公開鍵, $(E, W, KDF, hLen)$
	s	PSEC 秘密鍵, 非負整数, $0 \leq s < p$

処理手順:

1. 乱数 $s \in \{0, \cdots, p-1\}$ を生成する.
2. $W = sP$ を計算する.
3. $PK = (E, W, KDF, hLen)$ と s を出力する.

5.2 EP-PSEC

EP-PSEC(PK, α) を次のように定義する.

入力: PK PSEC 公開鍵
α 非負整数である乱数, $0 \leq \alpha < p$
出力: Q E 上の点
C_1 E 上の点
前提条件: 公開鍵 PK は正しい.
処理手順:

1. $Q = \alpha W$ を計算する.
2. $C_1 = \alpha P$ を計算する.
3. (Q, C_1) を出力する.

5.3 DP-PSEC

DP-PSEC(PK, C_1, s) を次のように定義する.

入力: PK PSEC 公開鍵
C_1 E 上の点
s PSEC 秘密鍵, 非負整数, $0 \leq s < p$
出力: Q E 上の点
前提条件: 公開鍵 PK および秘密鍵 s は正しい.
処理手順:

1. $Q = sC_1$ を計算する.
2. Q を出力する.

6 鍵カプセル化メカニズム (Key encapsulation mechanisms)

鍵カプセル化メカニズムは, 暗号化関数の入力が受信者の公開鍵のみである点を除いて, 公開鍵暗号のように働く. 暗号化関数は, (k, c_0) の対を生成する. k は決められた長さのビット列であり, c_0 は k を暗号化したものである. 復号関数は, c_0 から k を出力する.

ランダムなオクテット列を生成し, 受信者の公開鍵で暗号化する, という方法によっても通常の公開鍵暗号方式を用いて上記の目的を達成できるが, 以下に示すように, より効率良く鍵カプセル化メカニズムを実現することができる.

鍵カプセル化メカニズム PSEC-KEM は 2 つの処理から構成される.

- 暗号化処理 ES-PSEC-KEM-Encrypt(PK)
  入力は公開鍵 PK, 出力は鍵と暗号文の対 (k, c_0).
- 復号処理 ES-PSEC-KEM-Decrypt(PK, s, c_0)
  入力は公開鍵 PK, 秘密鍵 s および暗号文 c_0, 出力は鍵 k.

6.1 ES-PSEC-KEM

6.1.1 暗号化処理

ES-PSEC-KEM-Encrypt(PK) を次のように定義する.

入力: PK PSEC 公開鍵
出力: c_0 オクテット列
k オクテット列
前提条件: 公開鍵 PK は正しい.
処理手順:

1. (α, k, r) = EME-PSEC-KEM-A(PK) を求める. (7.1.1 章を見よ)
2. (Q, C_1) = EP-PSEC(PK, α) を求める. (5.2 章を見よ)
3. c_0 = EME-PSEC-KEM-B(PK, Q, C_1, r) を求める. (7.1.2 章を見よ)
4. (c_0, k) を出力する.

6.1.2 復号処理

ES-PSEC-KEM-Decrypt(PK, s, c_0) を次のように定義する.

入力: PK PSEC 公開鍵
s PSEC 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"
前提条件: 公開鍵 PK および秘密鍵 s は正しい.
処理手順:

1. (C_1, c_2, g) = EME-PSEC-KEM-C(PK, c_0) を求める. (7.1.3 章を見よ)
   デコーディング処理が "invalid" を返したならば, "invalid" をエラー出力して処理を終了する.
2. Q' = DP-PSEC(PK, C_1, s) を求める. (5.3 章を見よ)

3. (α', k')= EME-PSEC-KEM-D(PK, c_2, g, Q') を求める. (7.1.4 章を見よ)
   デコーディング処理が “invalid” を返したならば, “invalid” をエラー出力して処理を終了する.

4. $C_1 =$ DP-PSEC(PK, P, α') かどうかを検証する. (5.3 章を見よ)
   もし成立なら k' を出力し, 成立しないなら “invalid” をエラー出力して処理を終了する.

7 エンコーディングメソッド (Encoding methods)

本章では, 1 種類のエンコーディングメソッドを定義する.

7.1 EME-PSEC-KEM

7.1.1 エンコーディング処理 EME-PSEC-KEM-A

EME-PSEC-KEM-A(PK) を次のように定義する.

オプション:	$keyLen$	非負整数
入力:	PK	PSEC 公開鍵
出力:	α	非負整数, $0 \leq \alpha < p$
	k	オクテット列
	r	オクテット列

処理手順:

1. ランダムなオクテット列 $r \in \{0, \cdots, 255\}^{\lceil hLen/8 \rceil}$ を生成する.
2. $H =$ OS2BSP$(KDF(\text{I2OSP}(0, 32) \,||\, r, pLen + 128 + keyLen), pLen + 128 + keyLen)$ を求める.
3. H をビット長が $pLen + 128$ の t と, ビット長が $keyLen$ の k' との連結 $H = t \,||\, k'$ と解析する.
4. $\alpha =$ BS2IP$(t, pLen + 128) \bmod p$ を求める.
5. $k =$ BS2OSP$(k', keyLen)$ を求める.
6. (α, k, r) を出力する.

7.1.2 エンコーディング処理 EME-PSEC-KEM-B

EME-PSEC-KEM-B(PK, Q, C_1, r) を次のように定義する.

オプション: R Compressed, Uncompressed, Hybrid のいずれか
入力: PK PSEC 公開鍵
Q E 上の点
C_1 E 上の点
r オクテット列
出力: c_0 オクテット列
処理手順:

1. $c_2 = r \oplus KDF(\mathrm{I2OSP}(1,32) \,||\, \mathrm{ECP2OSP}(C_1, qmLen) \,||\, \mathrm{ECP2OSP}(Q, qmLen), hLen)$ を求める.
2. $c_0 = \mathrm{ECP2OSP}(C_1, qmLen) \,||\, c_2$ を求める.
3. c_0 を出力する.

7.1.3 デコーディング処理 EME-PSEC-KEM-C

EME-PSEC-KEM-C(PK, c_0) を次のように定義する.

オプション: R Compressed, Uncompressed, Hybrid のいずれか
入力: PK PSEC 公開鍵
c_0 オクテット列
出力: C_1 E 上の点
c_2 オクテット列
g 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 $\lceil hLen/8 \rceil$ 以下なら, “invalid” をエラー出力して処理を終了する.
2. c_0 を g と, オクテット長が $\lceil hLen/8 \rceil$ の c_2 の連結 $c_0 = g \,||\, c_2$ と解析する.
3. $C_1 = \mathrm{OS2ECPP}(g, qmLen)$ を求める.
4. OS2ECPP が “invalid” を返してきたなら, “invalid” をエラー出力して処理を終了する.
5. (C_1, c_2, g) を出力する.

7.1.4 デコーディング処理 EME-PSEC-KEM-D

EME-PSEC-KEM-D(PK, c_2, g, Q') を次のように定義する.

オプション:	$keyLen$	非負整数
入力:	PK	PSEC 公開鍵
	c_2	オクテット列
	g	オクテット列
	Q'	E 上の点
出力:	α'	非負整数, $0 \leq \alpha' < p$
	k'	オクテット列
エラー:	"invalid"	

処理手順:

1. $r' = c_2 \oplus KDF(\mathrm{I2OSP}(1, 32) \,||\, g \,||\, \mathrm{ECP2OSP}(Q', qmLen), hLen)$ を求める.
2. $h' = \mathrm{OS2BSP}(KDF(\mathrm{I2OSP}(0, 32) \,||\, r', pLen + 128 + keyLen), pLen + 128 + keyLen)$ を求める.
3. h' をビット長が $pLen + 128$ の t' と, ビット長が $keyLen$ の k'' との連結 $h' = t' \,||\, k''$ と解析する.
4. $\alpha' = \mathrm{BS2IP}(t', pLen + 128) \bmod p$ を求める.
5. $k' = \mathrm{BS2OSP}(k'', keyLen)$ を求める.
6. (α', k') を出力する.

8 補助関数 (Auxiliary techniques)

本章では, 本ドキュメントのさまざまな定義式に用いられる関数を説明する.

8.1 ハッシュ関数

本ドキュメントでは, SHA-1 ハッシュ関数を推奨する.

8.1.1 SHA-1

SHA-1 は FIPS PUB 180-1 [1] で定義されている. SHA-1 の出力長は 160 ビットであり, 処理ブロックサイズは 512 ビットである.

8.2 鍵導出関数

本ドキュメントでは, エンコーディングメソッドで用いられる鍵導出関数として, MGF1 マスク生成関数 [3] を推奨する.

MGF1 は [5] の中では KDF1 とも呼ばれている.

8.2.1 MGF1

MGF1 はハッシュ関数を利用したマスク生成関数である.
$MGF1(M, l)$ は次のように定義される.

オプション:	$Hash$	ハッシュ関数
	$hashLen$	ハッシュ関数の出力長, ビット長
入力:	M	オクテット列であるメッセージ
	l	出力の長さ, ビット長
出力:	$mask$	長さ $\left\lceil \frac{l}{8} \right\rceil$ のオクテット列
エラー:	"invalid"	

処理手順:

1. l_0 を M のビット長とする. もし, $l_0 + 32$ 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. $cThreshold = \left\lceil \frac{l}{hashLen} \right\rceil$ を計算する.
3. M' を空オクテット列とする.
4. $counter = 0$ とする.
   (a) counter を 32 ビット長のオクテット列に変換する.
   $$C = \mathrm{I2OSP}(counter, 32)$$
   (b) M と C を連結し, ハッシュ関数 $Hash$ を作用させる.
   $$H = Hash(M \mid\mid C)$$
   (c) M' と H を連結し, その結果を再び M' とする.
   $$M' = M' \mid\mid H$$
   (d) $counter$ を 1 増やす. もし $counter < cThreshold$ ならば, ステップ 4a に戻る.
5. M' の先頭 $\left\lceil \frac{l}{8} \right\rceil$ オクテットを, $mask$ とする.
   $$mask = M'_0 M'_1 \cdots M'_{\lceil l/8 \rceil - 1}$$
6. $mask$ を出力する.

参考文献

[1] FIPS PUB 180-1, “Secure Hash Standard (SHS),” U.S. Department of Commerce / National Institute of Standards and Technology, April 17, 1995.

[2] IEEE Std P1363-2000, “Standard Specifications for Public Key Cryptography: Additional Techniques,” draft version 8, IEEE, April 27, 2001.

[3] RSA Laboratories, “PKCS #1 v2.1: RSA Encryption Standard,” draft 2, January 5, 2001.

[4] Fujisaki, E. and Okamoto, T.: “Secure Integration of Asymmetric and Symmetric Encryption Schemes,” Proc. of Crypto’99, Springer-Verlag, LNCS 1666, pp. 535–554 (1999).

[5] Victor Shoup “A Proposal for an ISO Standard for Public Key Encryption,” http://shoup.net/papers/iso-2.pdf.

A 設計方針

PSEC-KEM は, 楕円 ElGamal 暗号関数（基本暗号プリミティブ）を [4] における変換方式で高い安全性を持つスキームに変換した（秘匿目的）暗号方式（PSEC-2 と呼んでいたスキーム）を鍵共有方式（Key Encapsulation Mechanism）に改良した方式 [5] （PSEC-2’ と呼ばれているもの）である.

PSEC-KEM は, 実用上様々な特長を持つ楕円曲線暗号関数（楕円 ElGamal 暗号関数）の利点を引き継ぐと共に, 最も高いレベルの安全性（適応的選択暗号文攻撃に対して強秘匿）をもつことが証明された暗号方式である.

B 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.

$$\begin{aligned} pLen &\geq 160 \\ hLen &\geq 128 \end{aligned}$$

C 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.

$$\begin{aligned} pLen &= 160 \\ KDF &= \text{MGF1 (SHA-1, } hashLen = 160) \\ hLen &= 160 \\ R &= \text{Compressed} \\ keyLen &= 256 \end{aligned}$$

D 高速実装法

PSEC-KEM を実装する場合, 以下の高速化技法を用いることができる.

- 楕円曲線演算を高速化する, 通常使える方法.
- 秘密鍵, 公開鍵, システムパラメータに関する事前演算処理を行なう方法.

E 実装上有用な情報

- 楕円曲線暗号のパラメータおよび実装法に関しては SECG などを参照せよ.
  (http://www.secg.org/)
- 多倍長演算は GMP などを使うことによって実装することができる.
  (http://www.swox.com/gmp/)

F バージョン情報

PSEC には PSEC-1, PSEC-2, PSEC-3 および PSEC-KEM の 4 種類がある. そのうち PSEC-2 と PSEC-3 はハイブリッド暗号である. PSEC-KEM は鍵カプセル化メカニズムであるが, ISO draft [5] 2.7 章のようにハイブリッド暗号として使うこともできる.

PSEC は, 以下の 4 つの応募または発表を行っている.

(a) ISO draft (PSEC-KEM)[5]

PSEC-2 をベースに, 鍵共有の機能を持つように変更した PSEC-2' が現在のドラフトに載っている. 名称を PSEC-KEM に変更する予定である. PSEC-2 との互換性はない.

推奨パラメータは特に決まっていない.

(b) NESSIE draft

2000 年に提出したバージョンは, PSEC-1,2,3 の プリミティブのみであり, エンコードメソッドは規定していない.

PSEC-1,3 を取り下げ, PSEC-2 を ISO ドラフトの PSEC-KEM と完全に互換性があるものに変更する予定である. 推奨パラメータは CRYPTREC 2001 と同じにする.

(c) CRYPTREC 2000

2000 年に提出したバージョンは, PSEC-1,2,3 の プリミティブのみであり, エンコードメソッドは規定していない.

推奨パラメータは, $pLen \geq 160, hLen \geq 128$.

(d) CRYPTREC 2001

PSEC-1,3 を取り下げ, 2000 年に応募した PSEC-2 を PSEC-KEM に名称を変更し, ISO ドラフトの PSEC-KEM と完全に互換性があるものに変更して継続応募する.

推奨パラメータは, $pLen \geq 160, hLen \geq 128$.

F.1 推奨パラメータ変更の理由

推奨パラメータの変更なし

F.2 互換性が異なる場合のユーザーの不利益

PSEC-1, PSEC-2, PSEC-3, および PSEC-KEM はエンコーディングメソッドが異なるため, スキームとして同一でない. したがって, 相互運用性が満たされない.