Specification of Data Types and Conversions version 1.0

NTT Secure Platform Laboratories,
NTT Corporation

March 2, 2015

1 Introduction

This document provides the specification of data operations using several different data types [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.
$\mathbf{R}$	the set of real numbers.
$\mathbf{Z}$	the set of integers.
$\mathbf{N}$	the set of positive integers.
$\mathbf{Z}_n$	the set of all integers x such that $0 \leq x < n$ and x is prime to n.
$a := b$	an operation of assigning b to a.
$\mathbf{F}_{p^m}$	a finite field with p^m elements, where p 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.
$\lceil y \rceil$	for $y \in \mathbf{R}$, the least integer greater than or equal to y.
$\lfloor y \rfloor$	for $y \in \mathbf{R}$, the greatest integer less than or equal to y.
$a \| b$	relation between integers a and b that holds if and only if a divides b, i.e., there exists an integer c such that $b = ac$.
$a \bmod m$	for $a \in \mathbf{Z}$, $m \in \mathbf{N}$, the least non-negative integer b that satisfies $m \| (a - b)$.
$a^{-1} \bmod m$	for $a \in \mathbf{Z}$, $m \in \mathbf{N}$, the least non-negative integer b that satisfies $ab \bmod m = 1$.
E	elliptic curve
$E(\mathbf{F}_p)$	the subgroup of $\mathbf{F}_p$-rational points.
$E(\mathbf{F}_{p^m})$	the subgroup of $\mathbf{F}_{p^m}$-rational points.
$E[q], E(\mathbf{F}_p)[q], E(\mathbf{F}_{p^m})[q]$	the subgroup of q-torsion point of $E, E(\mathbf{F}_p)$ and $E(\mathbf{F}_{p^m})$

2.2 Elliptic Curve Parameter

An elliptic curve parameter $\mathcal{E}$ in this document is specified by the following 8-tuple [2]

$$(\text{Curve-ID}, p_b, p_e, A, B, G = (x, y), q, h)$$

where the components have the following meanings:

- Curve-ID is an identifier with which the curve can be referenced.
- p_b is an odd prime specifying base field.
- p_e is an irreducible polynomial specifying extension field with degree m.
- A and B are the coefficients of the equation $y^2 = x^3 + Ax + B,\ (A, B \in \mathbf{F}_{q^m})$ defining E
- $G = (x, y)$ is the base point, i.e., a point with x and y being its x- and y-coordinates in E, respectively.

- q is the prime order of the group generated by G.
- h is a multiple of the cofactor of G in E.

In this document, we define h' as a multiple of h in addtion to 8-tuple. By using this h' in the hashing to point function, we can calculate the hashing to point function in high speed.

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

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

   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.

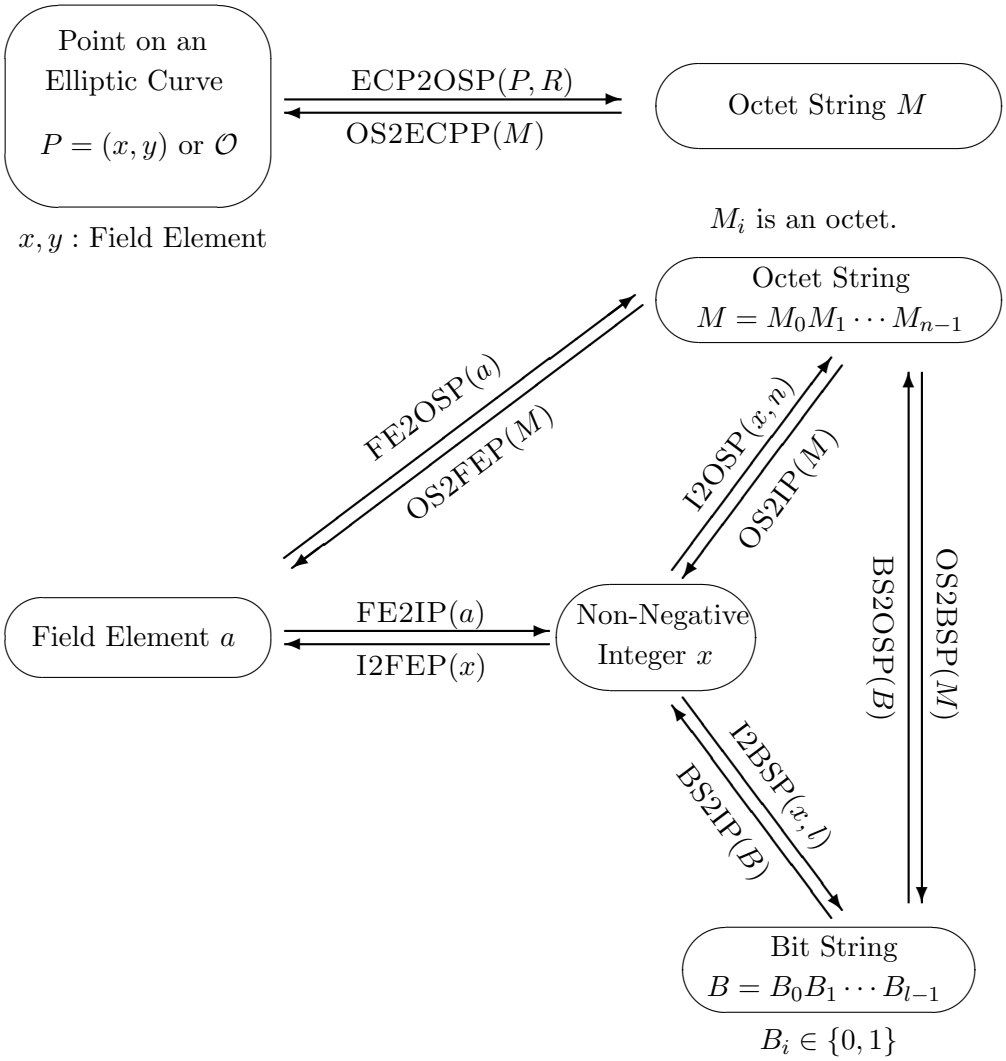

Figure 1: Conversions between data types

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. An finite field element should be represented as a polynomial with subfiled 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-p^m representation of the base field elements, where p^m is the number of the subfield elements. Formally, the conversion routine, FE2IP(a), is specified as follows:

System parameters:	$\mathbf{F}_{p^{m_2}} \supseteq \mathbf{F}_{p^{m_1}}$	a field extension with an irreduceble polynomial $Irr(\mathbf{F}_{p^{m_2}}/\mathbf{F}_{p^{m_1}}; \beta)$ i.e. $\mathbf{F}_{p^{m_2}} \simeq \mathbf{F}_{p^{m_1}}[\beta]/(Irr(\mathbf{F}_{p^{m_2}}/\mathbf{F}_{p^{m_1}}; \beta))$
Input:	a	a field element in $\mathbf{F}_{p^{m_2}}$
Output:	x	an integer in $\{0, \ldots, p^{m_2} - 1\}$
Steps:		

This Conversion procedure FE2IP is inductively defined. Convert field element a to integer x as follows:

If $m_2 = 1$ i.e. field $\mathbf{F}_{p^{m_2}}$ is prime field:

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

1. Let $x := a$.

2. Output x.

Else $m_2 > 1$:

For the field extension $\mathbf{F}_{p^{m_2}} \supseteq \mathbf{F}_{p^{m_1}}$, we assume that FE2IP is defined on $\mathbf{F}_{p^{m_1}}$ and outputs an integer in $\{0, \ldots, p^{m_1}\}$. Field element a must be represented as a polynomial of at most $(m_2/m_1 - 1)$-th degree with coefficients in the subfield $\mathbf{F}_{p^{m_1}}$. Let β be the variable of the polynomial.

1. Determine the coefficients $a_i \in \mathbf{F}_{p^{m_1}}$ for $i \in \{0, \ldots, m_2/m_1 - 1\}$ that satisfy

$$a = \sum_{i=0}^{m_2/m_1-1} a_i \beta^i.$$

2. Compute

$$x := \sum_{i=0}^{m_2/m_1-1} \mathrm{FE2IP}(a_i)(p^{m_1})^i.$$

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 subfield coefficients, and it can be represented as a sequence of the coefficients. Informally, the idea is to represent the integer with radix-p^m positional number system where p^m is the number of the subfield element, 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:	$\mathbf{F}_{p^{m_2}} \supseteq \mathbf{F}_{p^{m_1}}$	a field extension with an irreduceble polynomial $Irr(\mathbf{F}_{p^{m_2}}/\mathbf{F}_{p^{m_1}}; \beta)$ i.e. $\mathbf{F}_{p^{m_2}} \simeq \mathbf{F}_{p^{m_1}}[\beta]/(Irr(\mathbf{F}_{p^{m_2}}/\mathbf{F}_{p^{m_1}}; \beta))$
Input:	x	an integer in $\{0, \ldots, p^{m_2} - 1\}$
Output:	a	a field element in $\mathbf{F}_{p^{m_2}}$
Steps:		

This Conversion procedure I2FEP is inductively defined. Convert integer x to field element a as follows:

If $m_2 = 1$ i.e. $\mathbf{F}_{p^{m_2}}$ is prime field:

A field element of $\mathbf{F}_{p^{m_2}}$ must be represented as an integer in $\{0, \ldots, p - 1\}$.

1. Let $a := x$.
2. Output a.

Else $m_2 > 1$:

For the field extension $\mathbf{F}_{p^{m_2}} \supseteq \mathbf{F}_{p^{m_1}}$, we assume that I2FEP is defined on integers in $\{0, \ldots, p^{m_1} - 1\}$ and outputs an element in $\mathbf{F}_{p^{m_1}}$.

A field element of $\mathbf{F}_{p^{m_2}}$ must be represented as a polynomial of at most $(m_2/m_1 - 1)$-th degree with coefficients in $\mathbf{F}_{p^{m_1}}$. Let β be the variable of the polynomial.

1. Expand x into it's radix p^{m_1} representation $x_i \in \{0, \ldots, p^{m_1} - 1\}$ for $i \in \{0, \ldots, m_2/m_1 - 1\}$ that satisfies

$$x = \sum_{i=0}^{m_2/m_1-1} x_i (p^{m_1})^i.$$

2. Compute

$$a := \sum_{i=0}^{m_2/m_1-1} \text{I2FEP}(x_i)\beta^i.$$

3: Output a.

3.9 FieldElement-to-OctetString Conversion (FE2OSP)

The conversion routine, FE2OSP(a), is specified as follows:

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

1. Let

$$M := \text{I2OSP}(\text{FE2IP}(a), n).$$

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

1. Let

$$a := \text{I2FEP}(\text{OS2IP}(M)).$$

2. Output 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: $\mathcal{E}$ an elliptic curve parameter
Input: P a point on an elliptic curve over $\mathbf{F}_{p^m}$
R Compressed, Uncompressed, or Hybrid
Output: M an octet string of length n

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

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 $P = (x, y) \neq \mathcal{O}$ and $R =$ Compressed, proceed as follows:
   2.1. Set octet string $X := \text{FE2OSP}(x)$.
   2.2. Derive from 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 p is an odd number, set $\tilde{y} := 0$ if $y = 0$, otherwise set $\tilde{y} := y_i \bmod 2$, where $y = y_{m-1}\beta^{m-1} + \cdots + y_1\beta + y_0$, and i is the smallest integer such that $y_i \neq 0$.
   2.3. 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 $P = (x, y) \neq \mathcal{O}$ and $R =$ Uncompressed, proceed as follows:
   3.1. Set octet string $X := \text{FE2OSP}(x)$.
   3.2. Set octet string $Y := \text{FE2OSP}(y)$.
   3.3. Output $M := \text{I2OSP}(4, 1) \,\|\, X \,\|\, Y$.
4. If $P = (x, y) \neq \mathcal{O}$ and $R =$ Hybrid, proceed as follows:
   4.1. Set octet string $X := \text{FE2OSP}(x)$.
   4.2. Set octet string $Y := \text{FE2OSP}(y)$.
   4.3. Derive from y a single bit $\tilde{y}$ as follows (this allows the y-coordinate to be represented compactly using a single bit):
      4.3.1. Set $\tilde{y} := 0$ if $y = 0$, otherwise set $\tilde{y} := y_i \bmod 2$, where $y = y_{m-1}\beta^{m-1} + \cdots + y_1\beta + y_0$, and i is the smallest integer such that $y_i \neq 0$.
   4.4. 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:	$\mathcal{E}$	an elliptic curve parameter
Input:	M	an octet string that is either the single octet I2OSP$(0,1)$, an octet string of length $n = \left\lceil \frac{m \log_2 p}{8} \right\rceil + 1$, or an octet string of length $n = 2\left\lceil \frac{m \log_2 p}{8} \right\rceil + 1$.
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 $\left\lceil \frac{m \log_2 p}{8} \right\rceil$ octets X.
   2.2. Set $x := \text{OS2FEP}(X)$.
   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 x and $\tilde{y}$ elliptic curve point $P := (x, y)$, where:
      2.4.1. Compute the field element $w := x^3 + Ax + B$, and compute a square root γ of w in $\mathbf{F}_{p^m}$. Output `INVALID` and stop if there are no square roots in $\mathbf{F}_{p^m}$. Otherwise set $y = 0$ if $\gamma = 0$, otherwise set $y := \gamma$ if $\gamma_i \equiv \tilde{y} \bmod 2$, and set $y := -\gamma$ if $\gamma_i \not\equiv \tilde{y} \bmod 2$, where $\gamma = \gamma_{m-1}\beta^{m-1} + \cdots + \gamma_1\beta + \gamma_0$, and i is the smallest integer such that $\gamma_i \neq 0$.
   2.5. Output $P := (x, y)$.
3. If M has length $2\left\lceil \frac{m \log_2 p}{8} \right\rceil + 1$ octets, proceed as follows:
   3.1. Parse $M = L \,\|\, X \,\|\, Y$ as a single octet L followed by $\left\lceil \frac{m \log_2 p}{8} \right\rceil$ octets X followed by $\left\lceil \frac{m \log_2 p}{8} \right\rceil$ octets Y.
   3.2. Unless L is I2OSP$(4,1)$, I2OSP$(6,1)$ or I2OSP$(7,1)$, output `INVALID` and stop.
   3.3. Set $x := \text{OS2FEP}(X)$.

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

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

3.6. Output $P := (x, y)$.

3.13 Partial EllipticCurvePoint-to-OctetString Conversion (PECP2OSP)

The conversion routine, $\mathrm{PECP2OSP}(P)$, is specified as follows:

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

Steps:

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

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

4 Auxiliary

4.1 Key Derivation Functions (KDF)

As a key derivation function, the function described in Section 4.1.1 is recommended. The function is referred to as KDF1 in ISO/IEC 18033-2.

4.1.1 MGF1

MGF1 is a mask generation function, parameterized by a hash function. $\mathrm{MGF1}(M, n)$ 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*.

where, concrete *hash* function is SHA-256, SHA-384, SHA-512.

4.2 Group Membership Test

GroupMembershipTest$(\mathcal{E}, P)$ is a test function that an elliptic curve point is on the correct curve and group. GroupMembershipTest is defined as follows:

Input:	$\mathcal{E}$	an elliptic curve parameter
	$P = (x, y)$	an elliptic curve point
Output:	*bool*	an integer in $\{0, 1\}$
Error:	`INVALID`	
Steps:		

1. If $P = \mathcal{O}$, then output 1.
2. If P does not satisfy the elliptic curve equation i.e.

$$y^2 \neq x^3 + Ax + B,$$

output 0.

3. If $h \neq 1$ and $qP \neq \mathcal{O}$, then output 0.
4. Output 1.

4.3 Hashing to Point

4.3.1 IHF1

Bit strign should be converted to hashed non-negative integer less than an assainged integer as described in this section. Formally, the conversion routine, IHF1$(s, n, Hash)$ is defined as follows:

Input:	s	a bit string
	n	an integer
	$Hash$	a hash function
Output:	$v \in \mathbf{Z}_n$	
Steps:		

1. Set $hashLen$ be the length of the output of the hash funtion $Hash$.
2. Set h_0 be the zero string of length $hashLen$.
3. $h_1 := Hash(h_0||s)$.
4. $a_1 := \text{OS2IP}(h_1)$.
5. $h_2 := Hash(h_1||s)$.
6. $a_2 := \text{OS2IP}(h_2)$.
7. $v := 2^{hlen}a_1 + a_2 \mod n$.
8. Output v.

4.3.2 BS2FQE

Bit stirng should be converted to hashed finite field element as described in this section. Formally, the conversion routine, $\text{BS2FQE}(s, Hash, \mathbf{F}_{p^m})$ is defined as follows:

Input:	s	a bit string
	$Hash$	a hash function
	$\mathbf{F}_{p^m}$	a finite field with p^m elements where p is a prime, and $m > 0$ is an integer.
Output:	a	a element in $\mathbf{F}_{p^m}$
Steps:		

1. For $0 \leq i < m$,
   (a) $C := \text{I2OSP}(i, 2)$
   (b) $t_i := \text{IHF1}(C||s, p, Hash)$
2. $a := \sum_{i=0}^{m-1} t_i \beta^i$, where β is the variable of the polynomial.
3. Output a.

4.3.3 Hashing to Point

Hashed value should be converted to elliptic curve point as described in this section. Formally, the conversion routine, $\text{HashingToPoint}(\mathcal{E}, M)$, is specified as follows:

Input:	$\mathcal{E}$	an elliptic curve parameter.
	M	an octet string
Output:	P	an elliptic curve point
Steps:		

1. Let $i := 0$.

2. $C := \mathrm{I2OSP}(i, 2)$.
3. $x_0 := \mathrm{BS2FQE}(C||M, \mathit{Hash}, \mathbf{F}_{p^m}) \in \mathbf{F}_{p^m}$.
4. $t := x_0^3 + Ax_0 + B$.
5. If $t = 0$, then output $P = (x_0, 0)$.
6. If t is not square in $\mathbf{F}_{p^m}$, put $i := i + 1$ and return step 2.
7. Set $\pm\alpha$ are square roots of t.
8. Set $y_1 := \mathrm{FE2IP}(\alpha)$ and $y_2 := \mathrm{FE2IP}(-\alpha)$
9. If $y_1 > y_2$, then put $y_0 := -\alpha$. Else put $y_0 := \alpha$.
10. $P := (x_0, y_0)$.
11. Output $h'P$.

References

[1] IEEE Standard Specifications for Public-Key Cryptography, IEEE Std 1363-2000.

[2] K. Kasamatsu, S. Kanno, T. Kobayashi and Y. Kawahara: Barreto-Naehrig Curves draft-Kasamatsu-bncurves-01. Network Working Group Internet-Draft: (2014).