EPOC-2 Specification

NTT Information Sharing Platform Laboratories, NTT Corporation

October 12, 2001

Contents

1 Introduction

This document provides a specification for implementing EPOC-2, which is an encryption scheme. This document covers the following issues:

- cryptographic primitives: KGP-OU, EP-OU, DP-OU
- encryption schemes: ES-EPOC-2

This specification is compatible with the IFES-EPOC scheme in the IEEE P1363a draft [1], where the encryption and decryption primitives are IFEP-OU and IFDP-OU, respectively, and the message encoding method is EME3.

1.1 Overview

This document is organized as follows:

- Section 1 is an introduction.
- Section 2 defines some notations.
- Section 3 defines some data types and conversions.
- Section 4 defines the OU public and private keys used in KGP-OU, EP-OU, DP-OU.
- Section 5 defines several cryptographic primitives for the EPOC-2 encryption scheme.
- Section 6 defines the EPOC-2 encryption scheme (ES-EPOC-2).
- Section 7 defines the encoding method for the EPOC-2 encryption scheme.
- Section 8 defines some of the auxiliary functions used in this document.

2 Notation

N	the set of natural integers
$a := b$	give variable a the value of expression b
$(\mathbf{Z}/n\mathbf{Z})^*$	the set of all integers x such that $1 \leq x < n$ and x is prime to n.
$\{0,1\}^i$	the set of all bit strings of length i
$\{0,1\}^*$	$\bigcup_{i=0}^{\infty} \{0,1\}^i$
$\{0,1,\cdots,255\}^i$	the set of all octet strings of length i
$\{0,1,\cdots,255\}^*$	$\bigcup_{i=0}^{\infty} \{0,1,\cdots,255\}^i$
$\vert\vert$	a concatenation operator for two bit strings or a concatenation operator for two octet strings, for example, $(0,1,0,0)\,\vert\vert\,(1,1,0) = (0,1,0,0,1,1,0)$ for bit strings, $(4,3)\,\vert\vert\,(6,2) = (4,3,6,2)$ for octet strings
$\lceil y \rceil$	the least integer greater than or equal to y
$\lfloor y \rfloor$	the greatest integer less than or equal to y
a mod m	the least nonnegative integer b which satisfies $m \vert (b-a)$ for $a, m \in N$

The concatenation operator '$||$' is often omitted.

3 Data types and conversions

The schemes specified in this document involve operations using several different data types. Figure 1 illustrates which conversions are needed and where they are described.

3.1 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, l), is specified as follows:

Input:

B : a bit string of length l bits

l : an integer

Output:

M : an octet string of length $n = \lceil l/8 \rceil$ octets

Steps:

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

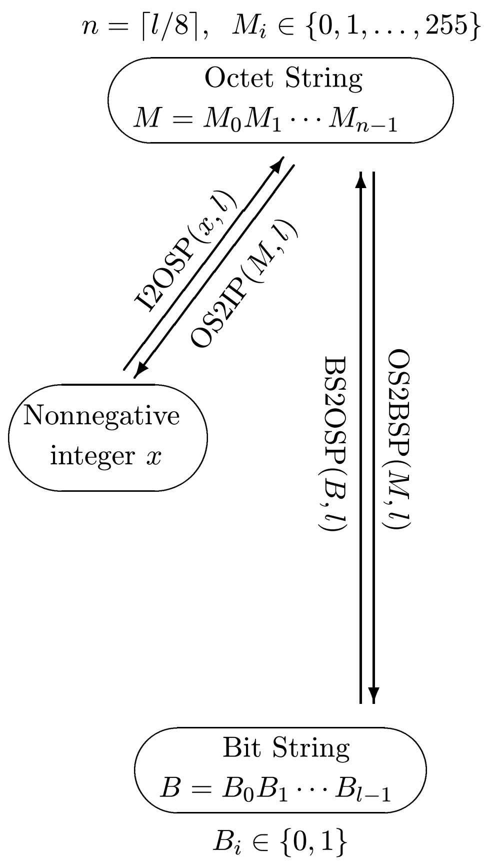
$n = \lceil l/8 \rceil, \ \ M_i \in \{0, 1, \ldots, 255\}$
Octet String
$M = M_0 M_1 \cdots M_{n-1}$
I2OSP(x,l)
OS2IP(M,l)
Nonnegative integer x
BS2OSP(B,l)
OS2BSP(M,l)
Bit String
$B = B_0 B_1 \cdots B_{l-1}$
$B_i \in \{0, 1\}$

Figure 1: Converting between data types

1. For $0 < i \leq n-1$, let:

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

2. Set the leftmost $8n-l$ bits in M_0 to 0's, and the rightmost $l+8-8n$ bits to $B_0 B_1 \ldots B_{l+7-8n}$.
3. Output M.

3.2 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, l), is specified as follows:

Input:

M	:	an octet string of length $n = \lceil l/8 \rceil$ octets
l	:	an integer

Output:

B	:	a bit string of length l bits

Steps:

Convert the octet string $M = M_0M_1 \ldots M_{n-1}$ to a bit string $B = B_0B_1 \ldots B_{l-1}$ as follows:

1. For $0 < i \leq n-1$, set:
$$B_{l-8-8(n-1-i)}B_{l-7-8(n-1-i)} \cdots B_{l-1-8(n-1-i)} := M_i.$$
2. Ignore the leftmost $8n - l$ bits of M_0, and set $B_0B_1 \ldots B_{l+7-8n}$ to the rightmost $l + 8 - 8n$ bits of M_0.
3. Output B.

3.3 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, l), is specified as follows:

Input:

x	:	a nonnegative integer
l	:	an integer

Output:

M	:	an octet string of length $n = \lceil l/8 \rceil$ octets

Errors:

"invalid"

Steps:

1. If $x \geq 2^l$, assert "invalid" and stop.
2. Determine the x's base-256 representation, $x_i \in \{0, \cdots, 255\}$ such that
$$x = x_{n-1}2^{8(n-1)} + x_{n-2}2^{8(n-2)} + \cdots + x_1 2^8 + x_0.$$
3. For $0 \leq i \leq n-1$, set $M_i := x_{n-1-i}$, and let
$$M := M_0M_1 \ldots M_{n-1}.$$
4. Output M.

3.4 OctetString-to-Integer Conversion(OS2IP)

Octet strings should be converted to integers as described in this section. Informally, the idea is simply to view the octet string as the base 256 representation of the integer. Formally, the conversion routine, OS2IP(M, l), is specified as follows:

Input:

M	:	an octet string of length $n = \lceil l/8 \rceil$ octets
l	:	an integer

Output:

x	:	an integer

Steps:

Convert $M = M_0 M_1 \ldots M_{n-1}$ to an integer, x, as follows:

1. View each M_i as an integer in $\{0, \ldots, 255\}$, and compute
$$x := \sum_{i=0}^{n-1} 2^{8(n-1-i)} M_i \mod 2^l.$$
2. Output x.

4 Key types

In this section, two types of keys are defined: OU public key and OU private key, both of which are used in three cryptographic primitives (KGP-OU, EP-OU, DP-OU) of EPOC-2.

4.1 OU public key

An OU public key is the 4-tuple $(n, g, h, pLen)$, where the components have the following meanings:

- n, the modulus, a nonnegative integer
- g, a nonnegative integer
- h, a nonnegative integer
- $pLen$, the security factor, a nonnegative integer

In a valid OU public key, modulus n takes the form $n = p^2 q$, where p and q are two distinct odd primes, and the bit length of p and q is $pLen$. g is an element in $(\mathbf{Z}/n\mathbf{Z})^*$ such that the order of g_p in $(\mathbf{Z}/p^2\mathbf{Z})^*$ is p, where $g_p := g^{p-1} \bmod p^2$. h is an element in $(\mathbf{Z}/n\mathbf{Z})^*$.

4.2 OU private key

An OU private key is the 4-tuple $(p, q, pLen, w)$, where the components have the following meanings:

- p, the first factor, a nonnegative integer
- q, the second factor, a nonnegative integer
- $pLen$, the security factor, a nonnegative integer
- w, the value of $L(g_p)$, where $L(x) = \frac{x-1}{p}$, a nonnegative integer

5 Cryptographic primitives

In this section, three cryptographic primitives are specified.

5.1 KGP-OU

KGP-OU(k) is defined as follows:

Input :	k	security parameter, a nonnegative integer
Output :	PK	OU public key $(n, g, h, pLen)$
	SK	OU private key $(p, q, pLen, w)$
Steps :		

1. Choose two primes p, q ($2^{k-1} \leq p < 2^k$, $2^{k-1} \leq q < 2^k$, $p \neq q$). Next, compute $n := p^2 q$.
2. Choose $g \in (\mathbf{Z}/n\mathbf{Z})^*$ such that the order of g_p in $(\mathbf{Z}/p^2\mathbf{Z})^*$ is p, where $g_p := g^{p-1} \bmod p^2$. (If $g_p \neq 1$, the order of g_p in $(\mathbf{Z}/p^2\mathbf{Z})^*$ is p.)
3. Let $h := g^n \bmod n$.
4. Set $pLen := k$.
5. Let $w := L(g_p)$, where $L(x) = \frac{x-1}{p}$.
6. Output $PK = (n, g, h, pLen)$, $SK = (p, q, pLen, w)$.

Remark : There are two typical ways to choose g (in step 2) as follows:

- **Example 1**
  Choose $g \in (\mathbf{Z}/n\mathbf{Z})^*$ randomly such that the order of g_p in $(\mathbf{Z}/p^2\mathbf{Z})^*$ is p (i.e., $g_p \neq 1$.)
- **Example 2**
  Choose $g = 2$. If the order of g_p in $(\mathbf{Z}/p^2\mathbf{Z})^*$ is not p (i.e., $g_p = 1$,) go back to step 1.

5.2 EP-OU

EP-OU(PK, m, r) is defined as follows:

Input :	PK	OU public key or a 4-tuple of integers $(n, g, h, pLen)$
	m	message representative, an integer, $0 \leq m < 2^{pLen-1}$
	r	random value, an integer, $0 \leq r < n$
Output :	c	ciphertext representative, an integer, $0 \leq c < n$
Errors :	"invalid"	
Steps :		

1. If the message representative m does not satisfy $0 \leq m < 2^{pLen-1}$, assert "invalid" and stop.

2. Let $c := g^m h^r \bmod n$.

3. Output c.

5.3 DP-OU

DP-OU(SK, c) is defined as follows:

Input :	SK	OU private key $(p, q, pLen, w)$
	c	ciphertext representative, an integer, $0 \leq c < p^2 q$
Output :	m	message representative, an integer, $0 \leq m < 2^{pLen-1}$
Errors :	"invalid"	
Assumptions :	private key SK is valid.	
Steps :		

1. If the ciphertext representative c does not satisfy $0 \leq c < p^2 q$, assert "invalid" and stop.

2. Let $c_p := c^{p-1} \bmod p^2$.

3. Let $m := \frac{L(c_p)}{w} \bmod p$, where $L(x) = \frac{x-1}{p}$.

4. If $0 \leq m < 2^{pLen-1}$, output m. Otherwise, assert "invalid" and stop.

6 EPOC-2 encryption scheme

In this section, the EPOC-2 encryption scheme (ES-EPOC-2) is specified.

6.1 ES-EPOC-2

6.1.1 Encryption operation

ES-EPOC-2-Encrypt(PK, M, P) is defined as follows:

Input :	PK	OU public key $(n, g, h, pLen)$
	M	message to be encrypted, an octet string
	P	encoding parameters, an octet string that may be empty
Output :	(C_1, C_2)	ciphertext, two octet strings
Errors :	"invalid"	
Assumptions :	public key PK is valid	
Steps :		

1. Let $(f, r, C_2) :=$ EME3-ENCODE$(M, pLen, P)$. (See Section 7.1.1.)
   If the encoding operation returns "invalid," then assert "invalid" and stop.

2. Let $C_1 :=$ EP-OU(PK, f, r). (See Section 5.2.)
   If the primitive returns "invalid," then assert "invalid" and stop.

3. Output the ciphertext (C_1, C_2).

6.1.2 Decryption operation

ES-EPOC-2-DECRYPT$(PK, SK, (C_1, C_2), P)$ is defined as follows:

Input :	PK	OU public key $(n, g, h, pLen)$
	SK	OU private key $(p, q, pLen, w)$
	(C_1, C_2)	ciphertext to be decrypted
	P	encoding parameters, an octet string that may be empty
Output :	M	message, an octet string
Errors :	"invalid"	
Steps :		

1. Let $f :=$ DP-OU(PK, SK, C_1). (See Section 5.3.)
   If the primitive returns "invalid," then assert "invalid" and stop.

2. Let $(M, r') :=$ EME3-DECODE$(C_2, f, pLen, P)$. (See Section 7.1.2.)
   If the decoding operation returns "invalid," then assert "invalid" and stop.

3. $PK' = (q, g \bmod q, h \bmod q, pLen)$.

4. If $C_1 \bmod q =$ EP-OU$(PK', f, r' \bmod (q-1))$, output M. Otherwise assert "invalid" and stop. (See Section 5.2.)

7 Encoding methods

In this section, one encoding method is specified.

7.1 EME3

7.1.1 Encoding operation

EME3-ENCODE$(M, pLen, P)$ is defined as follows:

Options :	$Hash$	hash function
	MGF	mask generation function
	KDF	key derivation function
	$SymEnc$	encryption function of symmetric cipher
	$hLen$	the bit length of the MGF output
	$oLen$	the bit length of the key of $SymEnc$
Input :	M	message to be encoded, an octet string
	$pLen$	the security factor, a nonnegative integer
	P	encoding parameters, an octet string that may be empty
Output :	f	the message representative, a nonnegative integer
	r	the random value, a nonnegative integer
	C_2	one of the ciphertexts, an octet string
Errors :	"invalid"	

Steps :

1. Let $rLen := \lfloor (pLen - 1)/8 \rfloor$.
2. Generate a random octet string $R \in \{0, \cdots, 255\}^{rLen}$.
3. Let $C_2 := SymEnc(KDF(R, oLen, P), M)$.
   If KDF returns "invalid," then assert "invalid" and stop.
4. Let $DB := M \,||\, R \,||\, C_2 \,||\, P$.
5. Let $H := MGF(Hash(DB), hLen)$.
   If $Hash$ returns "invalid," then assert "invalid" and stop.
6. Let f :=OS2IP(R).
7. Let r :=OS2IP(H).
8. Output (f, r, C_2).

Note:

- $Hash$ shall be one of the hash functions in Section 8.1.
- MGF shall be one of the mask generation functions in Section 8.2.
- KDF shall be one of the key derivation functions in Section 8.3.
- $SymEnc$ shall be one of the encryption function of symmetric ciphers in Section 8.4.

7.1.2 Decoding operation

EME3-Decode($C_2, f, pLen, P$) is defined as follows:

Options :	$Hash$	hash function
	MGF	mask generation function
	KDF	key derivation function
	$SymDec$	decryption function of symmetric cipher
	$hLen$	the bit length of the MGF output
	$oLen$	the bit length of the key of $SymDec$
Input :	C_2	one of the ciphertexts, an octet string
	f	the message representative, a nonnegative integer
	$pLen$	the security factor, a nonnegative integer
	P	encoding parameters, an octet string that may be empty
Output :	M	decoded message, an octet string
	r'	the random value, a nonnegative integer
Errors :	"invalid"	
Steps :		

1. Let $rLen := \lfloor (pLen - 1)/8 \rfloor$.
2. If $f \geq 256^{rLen}$, assert "invalid" and stop.
3. Let $R :=$ I2OSP$(f, rLen)$.
4. Let $M := SymDec(KDF(R, oLen, P), C_2)$
   If KDF returns "invalid," then assert "invalid" and stop.
5. Let $DB := M \,||\, R \,||\, C_2 \,||\, P$.
6. Let $H := MGF(Hash(DB), hLen)$
   If $Hash$ returns "invalid," then assert "invalid" and stop.
7. Let $r' :=$ OS2IP(H).
8. Output (M, r').

Note:

- $Hash$ shall be one of the hash functions in Section 8.1.
- MGF shall be one of the mask generation functions in Section 8.2.
- KDF shall be one of the key derivation functions in Section 8.3.
- $SymDec$ shall be one of the decryption function of symmetric ciphers in Section 8.4.

8 Auxiliary techniques

This section gives several examples of the techniques that support the functions described in this document.

8.1 Hash functions

One hash function is recommended for the encoding methods in this document: SHA-1.

8.1.1 SHA-1

SHA-1 is defined in FIPS PUB 180-1 [2]. The output length of SHA-1 is 160 bits, and the operation block size is 512 bits.

8.2 Mask generation functions

One mask generation function is recommended for the encoding methods in this document: MGF1 [3].

8.2.1 MGF1

MGF1 is a mask generation function based on a hash function.

MGF1(M, l) is defined as follows:

Options:	$Hash$	hash function
	$hashLen$	length in bits of the hash function output
Input:	M	seed from which mask is generated, an octet string
	l	intended bit length of the mask
Output:	$mask$	mask, an octet string of length $\left\lceil \frac{l}{8} \right\rceil$ octets
Errors:	"invalid"	
Steps:		

1. Let l_0 be the bit length of M. If $l_0 + 32$ is greater than the input limitation for the hash function, assert "invalid" and stop.
2. Let $cThreshold := \left\lceil \frac{l}{hashLen} \right\rceil$.
3. Let M' be the empty octet string.
4. Let $counter := 0$.
   (a) Convert the integer $counter$ to an octet string of length 32 bits:
   $$C := \mathrm{I2OSP}(counter, 32).$$

(b) Concatenate M and C, and apply the hash function to the result to produce a hash value:

$$H := Hash(M \mid\mid C).$$

(c) Concatenate M' and H to octet string M':

$$M' := M' \mid\mid H.$$

(d) Let $counter := counter + 1$. If $counter < cThreshold$, go back to step 4a.

5. Let $mask$ be the leftmost $\left\lceil \frac{l}{8} \right\rceil$ octets of the octet string M':

$$M'_0 M'_1 \cdots M'_{\lceil l/8 \rceil - 1}.$$

6. Output $mask$.

8.3 Key derivation functions

One key derivation function is recommended for the encoding methods in this document: KDF2 [1].

8.3.1 KDF2

KDF2 is a key derivation function based on a hash function.

KDF2(M, l, P) is defined as follows:

Options:	$Hash$	hash function
	$hashLen$	length in bits of the hash function output
Input:	M	seed from which key is generated, an octet string
	l	intended bit length of the key
	P	encoding parameters, an octet string that may be empty
Output:	key	key, an octet string of length $\left\lceil \frac{l}{8} \right\rceil$ octets
Errors:	"invalid"	

Steps:

1. Let l_0 be the bit length of M. If $l_0 + 32$ is greater than the input limitation for the hash function, assert "invalid" and stop.

2. Let $cThreshold := \left\lceil \frac{l}{hashLen} \right\rceil$.

3. Let M' be the empty octet string.

4. Let *counter* := 1.

   (a) Convert the integer *counter* to an octet string of length 32 bits:

   $$C := \mathrm{I2OSP}(counter, 32).$$

   (b) Concatenate M, C and P, and then apply the hash function to the result to produce a hash value:

   $$H := Hash(M \mid\mid C \mid\mid P).$$

   (c) Concatenate M' and H to the octet string M':

   $$M' := M' \mid\mid H.$$

   (d) Let $counter := counter + 1$. If $counter < cThreshold$, go back to step 4a.

5. Let *key* be the leftmost $\left\lceil \frac{l}{8} \right\rceil$ octets of the octet string M':

   $$M'_0 M'_1 \cdots M'_{\lceil l/8 \rceil - 1}.$$

6. Output *key*.

8.4 Encryption and decryption functions of symmetric ciphers

Two types of symmetric ciphers are recommended for the encoding methods in this document: Camellia [4] and OTP (one-time-pad) [5].

8.4.1 Camellia

Camellia is defined in NESSIE [4]. The key size of Camellia is 128, 192 or 256 bits, and the operation block size is 128 bits.

8.4.2 OTP

OTP (one-time-pad) is defined in Handbook of Applied Cryptography [5]. The key size of OTP is the same as the plaintext or ciphertext to be encrypted or decrypted.

References

[1] IEEE Std P1363a, "Standard Specifications for Public Key Cryptography: Additional Techniques," draft version 10 to be appeared, IEEE.

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

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

[4] NESSIE, http://cryptonessie.org/.

[5] Alfred John Menezes and Paul C. van Oorschot and Scott A. Vanstone, "Handbook of applied cryptography," CRC Press, 1997.

A Security requirements of parameters

Security requirements of EPOC-2 parameters are the following:

$$\begin{aligned} pLen &\geq 342 \qquad \text{(bit size of } n \geq 1024\text{)} \\ hLen &\geq 2pLen + 32 \end{aligned}$$

B Recommended values of parameters

Recommended values of EPOC-2 parameters are the following:

$$\begin{aligned} pLen &= 384 \qquad \text{(bit size of } n = 1152\text{)} \\ Hash &= \text{SHA-1} \\ MGF &= \text{MGF1 (SHA-1, } hashLen = 160\text{)} \\ KDF &= \text{KDF2 (SHA-1, } hashLen = 160\text{)} \\ SymEnc &= \text{Camellia (CBC, IV=0, key length=128)} \\ SymDec &= \text{Camellia (CBC, IV=0, key length=128)} \\ hLen &= 960 \\ oLen &= 128 \end{aligned}$$