Pseudorandom Number Generator *Enocoro*

Specification Ver. 2.0

Hitachi, Ltd.

2 February 2010

Copyright ©2009-2010 Hitachi, Ltd. All rights reserved.

Contents

Copyright ©2009-2010 Hitachi, Ltd. All rights reserved.

1 Introduction

This document provides the specification of *Enocoro* which is a pseudorandom number generator (PRNG) for a stream cipher.

1.1 History

Enocoro is a family of PRNGs which has 11 parameters. The common specification of *Enocoro* firstly published in [2], which is referred to as *Enocoro* v1 in this document. [2] recommended a set of parameters for 80-bit security and another set of parameters for 128-bit security, which are referred to as *Enocoro*-80v1 and *Enocoro*-128v1. Later, the recommended set of parameters for 128-bit security was changed (so *Enocoro*-128v1 is obsoleted) in [3] and it is referred to as *Enocoro*-128v1.1.

The common part of pseudorandom number generation algorithm described in this document is slightly different from *Enocoro* v1 and we refer to the updated common algorithm as *Enocoro* v2[1]. In addition, we define a new concrete algorithm for 128-bit security and we call it *Enocoro*-128v2. Enocoro-128v2 and Enocoro-128v1.1 differ only in the characteristic polynomial φ_8 over the finite field $\mathrm{GF}(2^8)$ and in the initialization process[2].

1.2 Organization of the document

First of all, the notations are defined and a number of mathematical concepts are explained in Section 2. The specification of the common part of the algorithm *Enocoro* v2 is given in Section 3. Then the recommended set of parameters and the initialization function is defined in Section 4.

2 Preliminaries

In this section we give some notations and knowledge of the underlying mathematical constructions.

[1] The difference of *Enocoro* v1 and *Enocoro* v2 is only their characteristic polynomials of $\mathrm{GF}(2^8)$.

[2] See [4] for the reasons for the change of the specification.

Copyright ©2009-2010 Hitachi, Ltd. All rights reserved.

2.1 Notations

$\oplus$	Bitwise XOR operation
$\wedge$	Bitwise AND operation
$\|\|$	Concatenation of two strings
$\ggg_m n$	Rotation n bits to the right (A m-bit register is expected)
$\lll_m n$	Rotation n bits to the left (A m-bit register is expected)
`0x`	Hexadecimal prefix

2.2 Data Structure

The elemental data size of *Enocoro* is 8-bit, namely a byte.

2.2.1 Data Representation

Enocoro uses operations defined over finite fields GF(2^8) and GF(2^4). The elements of a binary extension field is defined by a polynomial whose coefficients are 0 or 1. A polynomial is represented by a bit string. For example, the bit string `0x2` corresponds to the monomial x. We dealt with only GF(2^8) in this section for simple discussion. An element of GF(2^8) is given by a polynomial of degree less than 8. Such a polynomial $b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0$ is represented by $b_7||b_6||b_5||b_4||b_3||b_2||b_1||b_0$, where b_j is 0 or 1. For example, the polynomial $x^6 + x^4 + x^2 + x + 1$ is represented by `0x57 = 01010111`.

2.2.2 Addition

The sum of two polynomials over a finite field is the polynomial whose coefficients are given by the sum of corresponding coefficients modulo 2. In other words the addition is calculated by bitwise XOR of two bit strings. For example, the sum of `0x57` and `0xa3` is calculated as follows:

$$\begin{aligned}
\texttt{0x57} + \texttt{0xa3} &= (x^6 + x^4 + x^2 + x + 1) + (x^7 + x^5 + x + 1) \\
&= x^7 + x^6 + x^5 + x^4 + x^2 \\
&\leftrightarrow \texttt{0xf4}.
\end{aligned}$$

Copyright ©2009-2010 Hitachi, Ltd. All rights reserved.

2.2.3 Multiplication

In order to fix the multiplication rule, a characteristic polynomial φ_8 of degree 8 is firstly defined. In the specification of *Enocoro* v2, the following polynomial is used:

$$\varphi_8(x) = x^8 + x^4 + x^3 + x^2 + 1.$$

The bit string corresponds to $\varphi_8(x)$ is 0x11d[3].

The multiplication of the polynomial $f(x) = \sum a_i x^i$ by x is defined by

$$x \cdot f(x) = \sum a_i x^{i+1} \bmod \varphi_8(x).$$

For example,

$$\begin{aligned}
\texttt{0x02} \cdot \texttt{0x87} &= x \cdot (x^7 + x^2 + x + 1) \\
&= x^8 + x^3 + x^2 + x \\
&= (x^4 + x^3 + x^2 + 1) + (x^3 + x^2 + x) \\
&= x^4 + x + 1 \\
&= \texttt{0x13}.
\end{aligned}$$

The multiplication $f(x)$ by x^i for any positive integer i is defined by induction. The multiplication of any two elements $f(x) = \sum a_i x^i, g(x) = \sum b_i x^i$ is defined by

$$f \cdot g(x) = \sum_{i=0}^{14} \sum_{j=0}^{i} (a_j \wedge b_{i-j}) x^i \bmod \varphi_8(x).$$

2.2.4 Definition of GF(2^4)

Enocoro uses the multiplication over GF(2^4) as well as that over GF(2^8). The representation of the elements and the operations are defined in the similar manner to GF(2^8). An element of GF(2^4) is represented by 4-bit string $b_3||b_2||b_1||b_0$, which corresponds to the polynomial $b_3x^3 + b_2x^2 + b_1x + b_0$. The characteristic polynomial φ_4 for the finite field GF(2^4) is given by

$$\varphi_4(x) = x^4 + x + 1.$$

[3] *Enocoro* v1 uses $\varphi_8(x) = x^8 + x^4 + x^3 + x + 1$.

Copyright ©2009-2010 Hitachi, Ltd. All rights reserved.

2.3 Pseudorandom Number Generator

A PRNG consists of a finite state machine (FSM), an initialization function *Init*, and an output function *Out*. A FSM consists of a internal state (or a register) $S^{(t)}$ depending on the clock and its update function *Next*. The initialization function generates the initial internal state $S^{(0)}$ from the initial inputs such as a secret key K and an initial vector I. The output function generates output bits $Z^{(t)}$ from the internal state $S^{(t)}$ at each time t.

$$\begin{aligned} S^{(0)} &= \mathit{Init}(K, I), \\ Z^{(t)} &= \mathit{Out}(S^{(t)}), \\ S^{(t+1)} &= \mathit{Next}(S^{(t)}). \end{aligned}$$

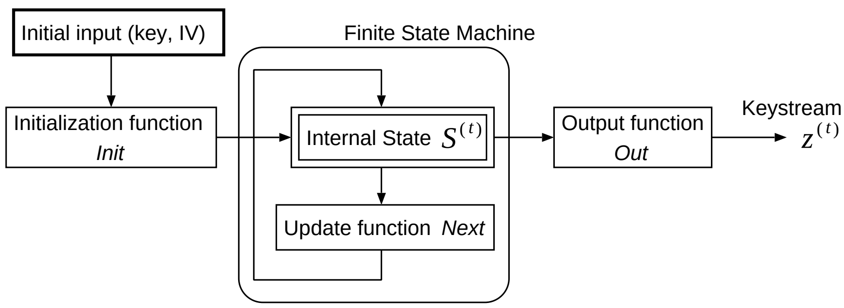

Figure 1: Pseudorandom Number Generator

2.4 Panama-like Keystream Generator

A Panama-like keystream generator (PKSG) is a class of PRNGs and is a generalization of software oriented PRNG Panama [1]. The internal state of a PKSG is separated into two: a *state* $a^{(t)}$ and a *buffer* $b^{(t)}$. The update functions are denoted by ρ and λ respectively and both functions take the other sub-internal state as a parameter. The whole update function *Next* is a composition of ρ and λ.

$$(a^{(t+1)}, b^{(t+1)}) = \mathit{Next}(S^{(t)}) = (\rho(a^{(t)}, b^{(t)}), \lambda(a^{(t)}, b^{(t)})).$$

Copyright ©2009-2010 Hitachi, Ltd. All rights reserved.

3 Common Specification of *Enocoro* v2

In this section, the specification of a family of PKSG *Enocoro* v2 is given. *Enocoro* v2 has 11 parameters. Let the buffer size of *Enocoro* v2 in byte be n_b, the inputs from the buffer to the ρ function be $b_{k_1}, b_{k_2}, b_{k_3}, b_{k_4}$. The parameters which define the λ function are denoted by $q_1, p_1, q_2, p_2, q_3, p_3$. It is denoted by $Enocoro(n_b; k_1, \ldots, k_4, q_1, p_1, \ldots, q_3, p_3)$ if the parameters are required to be explicitly described.

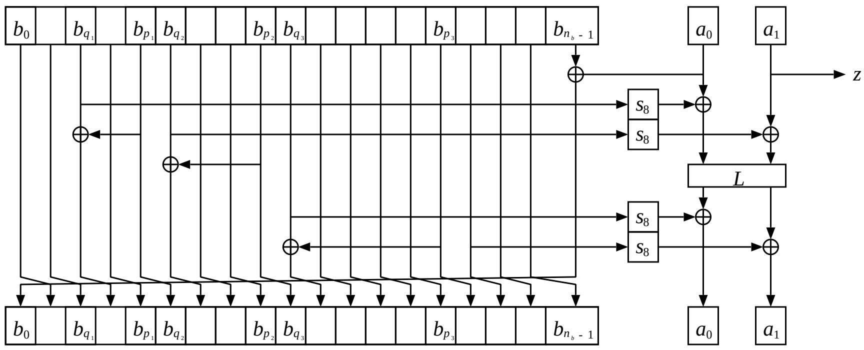

Figure 2: Schematic view of *Enocoro*

3.1 Internal State

The state a consists of two bytes. The higher byte is denoted by a_0 and the lower byte is denoted by a_1. The buffer b consists of n_b bytes. They are denoted by $b_0, b_1, \ldots, b_{n_b-1}$ in rotation.

3.2 Function ρ

The update function of the state ρ of *Enocoro* v2 takes $b_{k_1}, \ldots, b_{k_4}$ as external inputs. The ρ function consists of referring Sboxes, the linear transformation L defined over GF(2^8), and XORings. In detail, the transformation is defined

Copyright ©2009-2010 Hitachi, Ltd. All rights reserved.

as

$$
\begin{aligned}
u_0 &= a_0^{(t)} \oplus s_8[b_{k_1}^{(t)}], \\
u_1 &= a_1^{(t)} \oplus s_8[b_{k_2}^{(t)}], \\
(v_0, v_1) &= L(u_0, u_1), \\
a_0^{(t+1)} &= v_0 \oplus s_8[b_{k_3}^{(t)}], \\
a_1^{(t+1)} &= v_1 \oplus s_8[b_{k_4}^{(t)}].
\end{aligned}
$$

3.2.1 Linear Transformation

The transformation L of *Enocoro* v2 is chosen to be a linear transformation with a 2-by-2 matrix over $\mathrm{GF}(2^8)$, which is defined as

$$
\begin{pmatrix} v_0 \\ v_1 \end{pmatrix} = L(u_0, u_1) = \begin{pmatrix} 1 & 1 \\ 1 & d \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \end{pmatrix}, \quad d \in \mathrm{GF}(2^8).
$$

$d = \texttt{0x02}$ is adopted in *Enocoro* v2.

3.2.2 Sbox

The Sbox (substitution box) s_8 defines a permutation which maps 8-bit inputs to 8-bit outputs. It has also SPS structure and it consists of 4 small Sboxes s_4 which map 4-bit inputs to 4-bit outputs and a linear transformation l defined by a 2-by-2 matrix over $\mathrm{GF}(2^4)$. The Sbox s_4 is defined as

$$
s_4[16] = \{1, 3, 9, 10, 5, 14, 7, 2, 13, 0, 12, 15, 4, 8, 6, 11\}.
$$

The linear transformation l is defined as

$$
l(x, y) = \begin{pmatrix} 1 & e \\ e & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}, \quad x, y, e \in \mathrm{GF}(2^4)
$$

Figure 3 shows how to construct the 8-by-8 Sbox s_8. The Sbox s_8 is defined as

$$
\begin{aligned}
y_0 &= s_4[s_4[x_0] \oplus e \cdot s_4[x_1] \oplus \texttt{0xa}], \\
y_1 &= s_4[e \cdot s_4[x_0] \oplus s_4[x_1] \oplus \texttt{0x5}].
\end{aligned}
$$

$e = \texttt{0x04}$ is used for *Enocoro* v2. The output is rotated by 1 bit to the left at the end.

$$
s_8[x] = (y_0 || y_1) \lll_8 1.
$$

The table representation of the Sbox s_8 is given in Appendix.

Copyright ©2009-2010 Hitachi, Ltd. All rights reserved.

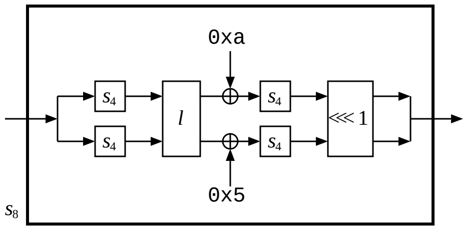

Figure 3: Sbox s_8

3.3 Function λ

The λ function of *Enocoro* consists of three feedbacks (XORings), XORing a_0 to the most right byte of the buffer b_{n_b-1}, and a byte-wise rotation of the buffer. In detail, the transformation is defined as follows:

$$\begin{aligned} b_i^{(t+1)} &= b_{i-1}^{(t)}, \quad i \neq 0, q_1+1, q_2+1, q_3+1, \\ b_0^{(t+1)} &= b_{n_b-1}^{(t)} \oplus a_0^{(t)}, \\ b_{q_j+1}^{(t+1)} &= b_{q_j}^{(t)} \oplus b_{p_j}^{(t)}, \quad j = 1, 2, 3, \end{aligned}$$

where $p_i - q_i \neq p_j - q_j$ if $i \neq j$.

3.4 Output function *Out*

The output function of *Enocoro* v2 outputs the lower byte of the state.

$$Out(S^{(t)}) = a_1^{(t)}.$$

3.5 Inputs and Initialization Function

The way to set inputs (a key and an IV) and the initialization function for each algorithm are defined in Section 4.2.

Copyright ©2009-2010 Hitachi, Ltd. All rights reserved.

4 *Enocoro*-128v2

4.1 Parameters

Enocoro-128v2 is a PRNG which takes a 128-bit key input, a 64-bit initial vector, and the following specified parameters:

$$\begin{aligned}
&n_b = 32,\\
&k_1 = 2, \quad k_2 = 7, \quad k_3 = 16, \quad k_4 = 29,\\
&p_1 = 6, \quad p_2 = 15, \quad p_3 = 28,\\
&q_1 = 2, \quad q_2 = 7, \quad q_3 = 16.
\end{aligned}$$

4.2 Initialization Function

Firstly, the initialization function sets to the registers a key K, an IV I and the initial constants C as follows:

$$\begin{aligned}
b_i^{(-96)} &= K_i, \quad 0 \le i < 16,\\
b_{i+16}^{(-96)} &= I_i, \quad 0 \le i < 8,
\end{aligned}$$

$$\begin{aligned}
b_{24}^{(-96)} &= C_0 = \texttt{0x66},\\
b_{25}^{(-96)} &= C_1 = \texttt{0xe9},\\
b_{26}^{(-96)} &= C_2 = \texttt{0x4b},\\
b_{27}^{(-96)} &= C_3 = \texttt{0xd4},\\
b_{28}^{(-96)} &= C_4 = \texttt{0xef},\\
b_{29}^{(-96)} &= C_5 = \texttt{0x8a},\\
b_{30}^{(-96)} &= C_6 = \texttt{0x2c},\\
b_{31}^{(-96)} &= C_7 = \texttt{0x3b},\\
a_0^{(-96)} &= C_8 = \texttt{0x88},\\
a_1^{(-96)} &= C_9 = \texttt{0x4c}.
\end{aligned}$$

Then the state is updated by the 96 iterations of two functions: one is an XORing of the counter to b_{31} and the other is the update function *Next*. The size of the counter is a byte. It is initialized by `0x01` and incremented

Copyright ©2009-2010 Hitachi, Ltd. All rights reserved.

by the multiplication by `0x02` which is defined over the finite field GF(2^8). In order to remove any ambiguity, we also define the initialization function as the following pseudo-code:

```
Init (a[2], b[32], K[16], I[8]){
  // set initial values
  for (i = 0; i < 16; i++){ b[i] = K[i]; }
  for (i = 0; i < 8; i++){ b[i+16] = I[i]; }
  for (i = 0; i < 8; i++){ b[i+24] = C[i]; }
  a[0] = C[8]; a[1] = C[9];
  ctr = 1;

  // update the state 96 times
  for (r = 0; r < 96; r++){
      b[31] ^= ctr;
      ctr = gf256multiplication(ctr, 2);
      Next(a, b);
  }
}
```

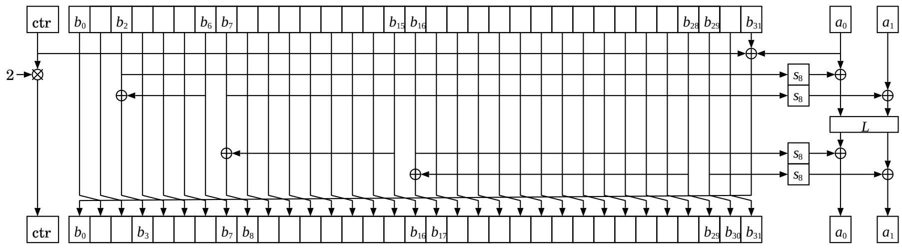

Figure 4: State update during the initialization of *Enocoro*-128v2

5 Data Encryption Using *Enocoro*-128v2

5.1 Choice of A Key and An Initial Vector

In general, the output sequence generated by any PRNGs is uniquely determined by the combination of the secret key K and the initial vector I. So it

Copyright ©2009-2010 Hitachi, Ltd. All rights reserved.

is not allowed to use an identical combination twice. Especially, in case that key streams are generated under the same key, different initial vectors must be used.

5.2 Encryption and Decryption

A binary additive mode provides a data encryption mechanism using a PRNG and it just combines the keystream and the plaintext by means of bitwise XORs. The decryption is done by the same manner. Let $p^{(t)}$, $c^{(t)}$, $z^{(t)}$ be the plaintext, the ciphertext, and the output byte at time t respectively. Then the (byte-wise) binary additive encryption and decryption are defined by

$$\begin{aligned} c^{(t)} &= p^{(t)} \oplus z^{(t)}, \\ p^{(t)} &= c^{(t)} \oplus z^{(t)}. \end{aligned}$$

References

[1] J. Daemen, C. Clapp, "Fast Hashing and Stream Encryption with PANAMA," *Fast Software Encryption, FSE'98*, Springer-Verlag, LNCS 1372, pp.60–74, 1998.

[2] D. Watanabe and T. Kaneko, "A construction of light weight Panama-like keystream generator," IEICE Technical report, ISEC2007-78, 2007 (*in Japanese*).

[3] K. Muto, D. Watanabe and T. Kaneko, "Strength evaluation of *Enocoro*-128 against LDA and its Improvement," *Symposium on Cryptography and Information Security, SCIS 2008*, 4A1-1, 2008 (*in Japanese*).

[4] D. Watanabe, K. Okamoto and T. Kaneko, "A Hardware-Oriented Light Weight Pseudorandom Number Generator *Enocoro*-128v2," *Symposium on Cryptography and Information Security, SCIS2010*, 3D1-3, 2010 (*in Japanese*).

Copyright ©2009-2010 Hitachi, Ltd. All rights reserved.

A Sbox s_8

The following array is the table representation of 8-bit Sbox s_8.
$s_8[256] = \{$

```
 99, 82, 26, 223, 138, 246, 174, 85, 137, 231, 208, 45, 189, 1, 36, 120,
27, 217, 227, 84, 200, 164, 236, 126, 171, 0, 156, 46, 145, 103, 55, 83,
78, 107, 108, 17, 178, 192, 130, 253, 57, 69, 254, 155, 52, 215, 167, 8,
184, 154, 51, 198, 76, 29, 105, 161, 110, 62, 197, 10, 87, 244, 241, 131,
245, 71, 31, 122, 165, 41, 60, 66, 214, 115, 141, 240, 142, 24, 170, 193,
32, 191, 230, 147, 81, 14, 247, 152, 221, 186, 106, 5, 72, 35, 109, 212,
30, 96, 117, 67, 151, 42, 49, 219, 132, 25, 175, 188, 204, 243, 232, 70,
136, 172, 139, 228, 123, 213, 88, 54, 2, 177, 7, 114, 225, 220, 95, 47,
93, 229, 209, 12, 38, 153, 181, 111, 224, 74, 59, 222, 162, 104, 146, 23,
202, 238, 169, 182, 3, 94, 211, 37, 251, 157, 97, 89, 6, 144, 116, 44,
39, 149, 160, 185, 124, 237, 4, 210, 80, 226, 73, 119, 203, 58, 15, 158,
112, 22, 92, 239, 33, 179, 159, 13, 166, 201, 34, 148, 250, 75, 216, 101,
133, 61, 150, 40, 20, 91, 102, 234, 127, 206, 249, 64, 19, 173, 195, 176,
242, 194, 56, 128, 207, 113, 11, 135, 77, 53, 86, 233, 100, 190, 28, 187,
183, 48, 196, 43, 255, 98, 65, 168, 21, 140, 18, 199, 121, 143, 90, 252,
205, 9, 79, 125, 248, 134, 218, 16, 50, 118, 180, 163, 63, 68, 129, 235
```

$\};$

Copyright ©2009-2010 Hitachi, Ltd. All rights reserved.

B Test Vectors

key[16] = {0}
iv[8] = {0}
output =
`0x63 0xd7 0xda 0x6b 0x55 0x73 0x7f 0xcf`
`0x57 0x34 0xb6 0x77 0x3a 0xe7 0x72 0xe8`
...

key[10] =
{`0x00`, `0x01`, `0x02`, `0x03`, `0x04`, `0x05`, `0x06`, `0x07`
`0x08`, `0x09`, `0x0a`, `0x0b`, `0x0c`, `0x0d`, `0x0e`, `0x0f`}
iv[8] =
{`0x00`, `0x10`, `0x20`, `0x30`, `0x40`, `0x50`, `0x60`, `0x70`}
output =
`0xc8 0xc8 0xee 0x43 0x3b 0x0d 0xc0 0x40`
`0xe5 0x3b 0xc5 0x06 0xea 0x21 0xad 0x82`
...

Copyright ©2009-2010 Hitachi, Ltd. All rights reserved.