

疑似乱数生成器 *Enocoro*

仕様書 Ver. 2.0

株式会社 日立製作所

2010年2月2日

目次

1	序	3
1.1	履歴	3
1.2	本稿の構成	3
2	準備	3
2.1	記号	4
2.2	データ構造	4
2.3	有限体上の演算	4
2.3.1	データの表現	4
2.3.2	元の加算	5
2.3.3	元の乗算	5
2.3.4	$GF(2^4)$ の定義	5
2.4	疑似乱数生成器	6
2.5	PANAMA 型生成器	6
3	<i>Enocoro v2</i> の共通仕様	7
3.1	内部状態	7
3.2	ρ 関数	7
3.2.1	線形変換	8
3.2.2	Sbox	8
3.3	λ 関数	9
3.4	出力関数 <i>Out</i>	9
3.5	初期入力と初期化関数	9
4	<i>Enocoro-128v2</i>	9
4.1	パラメータ	9
4.2	初期化	10
5	暗号処理への利用と注意	11
5.1	鍵と初期ベクトルの与え方	11
5.2	暗号化と復号	12
A	Sbox s_8	13
B	テストベクトル	14

1 序

本稿は、ストリーム暗号向けの疑似乱数生成器 *Enocoro* の仕様書である。

1.1 履歴

Enocoro は 11 個のパラメータを持つ疑似乱数生成用アルゴリズム群である。*Enocoro* の共通仕様は [1] で公開された。本稿では、この共通仕様を *Enocoro* ver.1 もしくは *Enocoro* v1 と呼ぶ。さらに、[1] では鍵長 80 ビットと鍵長 128 ビットの場合について、推奨パラメータと固有の初期化アルゴリズムが与えられている。本提案では、これら固有の疑似乱数生成器を必要に応じて *Enocoro*-80 ver.1.0 および *Enocoro*-128 ver.1.0 と呼ぶ。冗長さを取り除くため、*Enocoro*-80v1 と表す。その後、鍵長 128 ビットについては、推奨パラメータの変更が行われた [2]。このアルゴリズムを *Enocoro*-128 ver.1.1 または *Enocoro*-128 v1.1 と呼ぶ。

本稿で記述するアルゴリズムは、共通仕様が *Enocoro* v1 とはわずかに異なる^{*1}。本稿で記述するアルゴリズムを明示する必要がある場合には、*Enocoro* ver.2 もしくは *Enocoro* v2 と呼ぶ。また、本稿では、鍵長 128 ビットの場合について、*Enocoro* v2 の推奨パラメータと固有の初期化アルゴリズムを与える。この疑似乱数生成器を *Enocoro*-128 ver.2.0 もしくは *Enocoro*-128v2 と呼ぶ。ver.2.0 は ver.1.1 と同じパラメータを持つが、 $GF(2^8)$ の定義多項式 φ_8 および初期化関数が異なる^{*2}。

1.2 本稿の構成

本稿では、まず 2 章で仕様を理解するために必要な記号、基礎知識について簡単に説明する。次に 3 章で *Enocoro* v2 の仕様を詳述する。また、4 章で推奨パラメータを定める。

2 準備

この章では、仕様を記述するために必要な諸事項について解説する。

^{*1} *Enocoro* v1 と *Enocoro* v2 の違いは $GF(2^8)$ の定義多項式のみである。

^{*2} ver.1.1 から ver.2.0 への仕様変更の経緯は [3] を参照のこと。

2.1 記号

\oplus	ビットごとの排他的論理和
\parallel	二つのビット列の連結
$\ggg_m n$	右まわりで n ビット巡回シフト (レジスタ幅は m ビット)
$\lll_m n$	左まわりで n ビット巡回シフト (")
0x	整数の 16 進表記を意味する接頭語

2.2 データ構造

Enocoro では、1 バイトをデータの基本単位とする。

2.3 有限体上の演算

2.3.1 データの表現

本稿では、有限体 $\text{GF}(2^8)$ 上の演算を用いる。有限体の表現法は一意ではないので、この節で一つの表現を固定する。

まず、 $\text{GF}(2^8)$ の定義多項式 $\varphi_8(x)$ を

$$\varphi_8(x) = x^8 + x^4 + x^3 + x^2 + 1 = 0x11d,$$

で与える^{*3}。すなわち、 $\text{GF}(2^8) = \text{GF}(2)[x]/(\varphi_8(x))$ として定義する。

有限体の元は $\text{GF}(2)$ 上の (すなわち、 $\{0, 1\}$ を係数に持つ 7 次以下の) 1 変数多項式として定義される。データとしては、係数を降べきに並べた 8 ビットで一つの元を表現する。すなわち、ビット列 $b_7||b_6||b_5||b_4||b_3||b_2||b_1||b_0$ で多項式

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0,$$

を表現する。たとえば、0x57 はビット列 0101 0111 に対応するので、多項式 $x^6 + x^4 + x^2 + x + 1$ を表す。

^{*3} *Enocoro* v1 では、 $\varphi_8(x) = x^8 + x^4 + x^3 + x + 1$ を用いた。

2.3.2 元の加算

2つの多項式の加算は、同じ次数ごとに係数の mod 2 での加算、すなわち排他的論理和として定義される。例えば、

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

2.3.3 元の乗算

GF(2⁸) 上の乗算について、2つのステップに分けて説明する。

まず、GF(2⁸) の元 $f(x) = \sum a_i x^i$ と x との乗算 $x \cdot f(x)$ を

$$\sum a_i x^{i+1} \bmod \varphi_8(x)$$

で定義する。たとえば、

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

$x^i \cdot f(x)$ は、上記定義から帰納的に計算できる。

GF(2⁸) の任意の2つの元 $f(x) = \sum a_i x^i, g(x) = \sum b_i x^i$ の乗算 $f \cdot g$ は

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

で定義する。

2.3.4 GF(2⁴) の定義

本稿では、有限体 GF(2⁴) 上の演算も同様に用いる。元の表現、演算の定義自体は GF(2⁸) に定義できる。GF(2⁴) の元はビット列 $b_3||b_2||b_1||b_0$ で多項式 $b_3x^3 + b_2x^2 + b_1x + b_0$ を表現する。また、GF(2⁴) の定義多項式 φ_4 は次式で定義する。

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

2.4 疑似乱数生成器

疑似乱数生成器 (Pseudo-Random Number Generator: PRNG) は固定長の初期入力から任意長のビット列を生成する確定的アルゴリズムであり、有限状態生成器 (Finite State Machine: FSM)、初期化関数 $Init$ 、出力関数 Out からなる。有限状態生成器とは、時刻に依存する内部状態レジスタ $S^{(t)}$ と内部状態の更新関数 $Next$ の組で定義される。初期化関数は初期入力 (鍵、IV など) から内部状態レジスタの初期状態 $S^{(0)}$ を生成する関数である。また、出力関数は時刻 t の内部状態 $S^{(t)}$ から時刻 t の出力 $z^{(t)}$ を定める関数である。

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

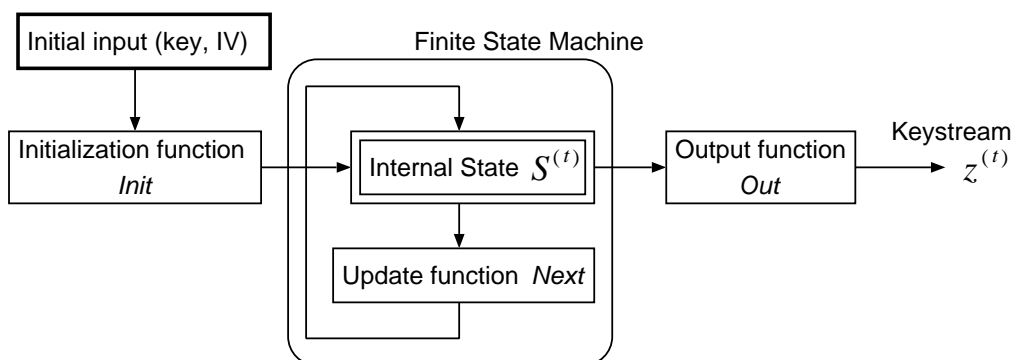


図 1 疑似乱数生成器

2.5 PANAMA 型生成器

PANAMA-like Key Stream Generator (PKSG) は、疑似乱数生成器の一種である。PKSG では、内部状態 $S^{(t)}$ を 2 つの部分 (ステート $a^{(t)}$ とバッファ $b^{(t)}$ と呼ぶ) に分け、それぞれの状態更新関数を ρ, λ とする。 ρ は $a^{(t)}$ の状態更新関数であり、 $b^{(t)}$ をパラメータとする。 λ は $b^{(t)}$ の状態更新関数であり、 $a^{(t)}$ をパラメータとする線形変換である。状態更新関数 $Next$ は 2 つの状態更新関数 ρ, λ の合成関数として記述される。

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

3 Enocoro v2 の共通仕様

Enocoro は PKSG である。バッファの大きさを n_b 、バッファからの ρ 関数への入力をパラメータ $b_{k_1}, b_{k_2}, b_{k_3}, b_{k_4}$ で、 λ 関数を定めるフィードバックをパラメータ $q_1, p_1, q_2, p_2, q_3, p_3$ で定義する。パラメータを明記する場合には $Enocoron_b; k_1, \dots, k_4, q_1, p_1, \dots, q_3, p_3$ と記述する。

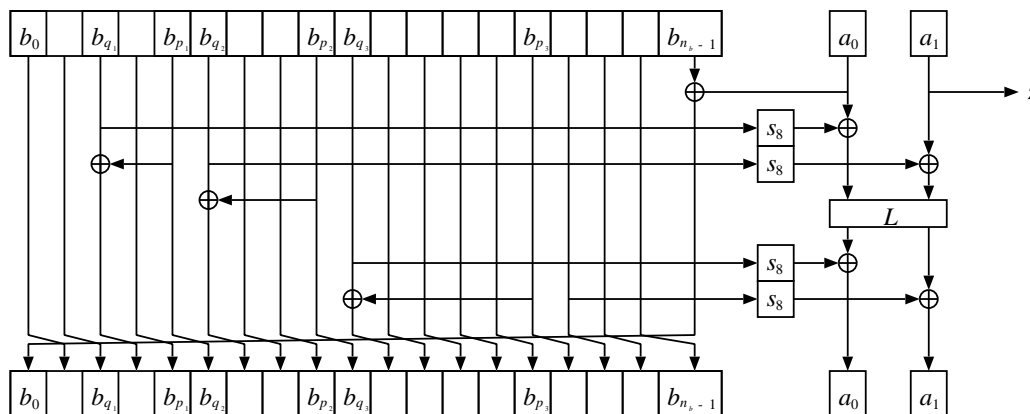


図 2 Enocoro 概要図

3.1 内部状態

状態 a は 2 バイトからなる。上位バイトから順に a_0, a_1 と表す。

バッファ b は n_b バイトからなる。上位バイトから順に $b_0, b_1, \dots, b_{n_b-1}$ と表す。

3.2 ρ 関数

Enocoro の ρ 関数は Sbox の参照と有限体 $GF(2^8)$ 上の 2×2 行列による線形変換 L 、排他的論理和で構成される。また、バッファの特定バイトの値 b_{k_1}, \dots, b_{k_4} をパラメータとして用いる。 ρ 関数の処理は以下の手順で行われる。

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

以下、要素演算 L , s_8 を記述する。

3.2.1 線形変換

線形変換 L は $\text{GF}(2^8)$ 上の 2×2 行列であり、以下の式で定義される。

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

の形の行列を用いる。特に、後述する *Enocoro-128v2* では $d = 0x02$ とする。

3.2.2 Sbox

まず、8ビット入力8ビット出力の Sbox s_8 の仕様を述べる。4ビット入力4ビット出力の Sbox s_4 を以下で定義する。

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

次に、以下の行列で定義される線形変換を用いて、SPS 変換を構成する。

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

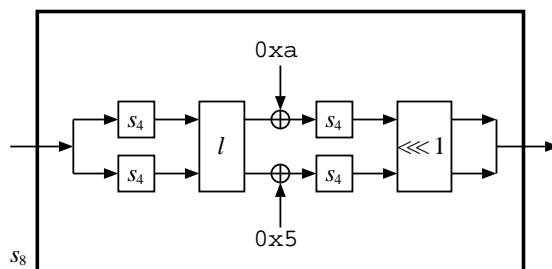


図3 Sbox s_8 の生成

SPS 変換の構成の概要は図 3 に従う。すなわち、

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

となる。Enocoro-128v2 では $e = 0x04$ を用いる。

最後に全体を 1 ビット巡回シフト演算を施し、有限体上の処理としての対称性を崩す。

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

付録 A に s_8 の配列を記載する。

3.3 λ 関数

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

ただし、 $i \neq j$ ならば $p_i - q_i \neq p_j - q_j$ とする。

3.4 出力関数 Out

出力関数は状態 a の下位バイトを出力する。

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

3.5 初期入力と初期化関数

初期化関数はアルゴリズムごとに記述する。

4 Enocoro-128v2

4.1 パラメータ

Enocoro-128v2 は鍵長 128 ビット、初期ベクトル長 64 ビットの疑似乱数生成器であ

る。Enocoro-128v2 のパラメータは以下のとおりである。

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

4.2 初期化

Enocoro-128v2 の初期化では、まず鍵 K 、IV I と初期定数 C を以下のようにレジスタにセットする。

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

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

その後、カウンタ値の排他的論理和と状態更新関数 $Next$ の適用を 96 回繰り返して、初期状態 $S^{(0)} = (a^{(0)}, b^{(0)})$ を生成する。カウンタ値はバッファの最下位バイト b_{31} に排他的論理和される。ただし、カウンタ値の初期値は 1 とし、インクリメントは $GF(2^8)$ 上での 2 倍算で定義する。初期化の擬似コードは以下で与えられる。

```
Init (a, b, K, I){
    // 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);
}
}

```

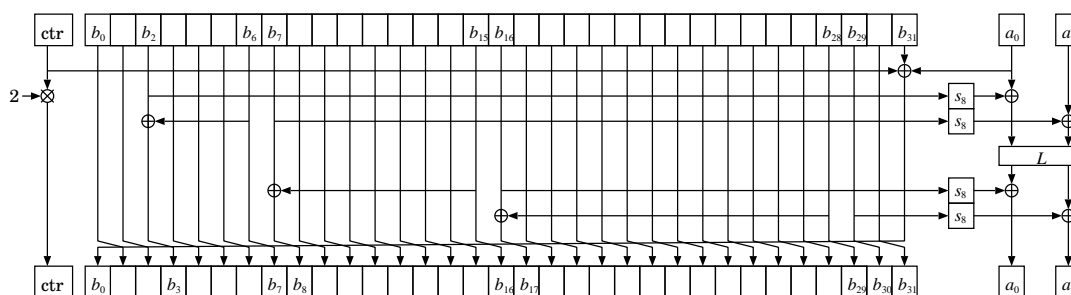


図4 Enocoro-128v2 の初期化における状態更新

5 暗号処理への利用と注意

5.1 鍵と初期ベクトルの与え方

一般に、確定アルゴリズムで定義される疑似乱数生成器の特性として、出力される乱数列は秘密鍵 K 、初期ベクトル I の組み合わせにより一意に決まる。したがって、まったく同じ組み合わせのものを用いてはならない。特に、同じ鍵 K を用いて複数の鍵ストリームを生成する場合には、異なる初期ベクトル I を用いる必要がある。

また、Enocoro-128v2 では、1組の鍵と初期ベクトルに対して 2^{64} バイトより大きな鍵ストリームを生成してはならない。

5.2 暗号化と復号

Enocoro-128v2 を用いてストリーム暗号を実現する場合には、平文データ P を 8 ビットずつのデータブロックに分割し、鍵 K 、初期ベクトル I を用いて生成した乱数列とブロックごとに排他的論理和すれば良い。復号化もまったく同様にして実現できる。

参考文献

- [1] 渡辺大, 金子敏信, “軽量の PANAMA 型疑似乱数生成器の構成に関する検討,” IEICE Technical report, ISEC2007-78, 2007.
- [2] 武藤健一郎, 渡辺大, 金子敏信, “Enocoro-128 の LDA 耐性評価と改良,” 暗号と情報セキュリティシンポジウム, SCIS2008, 4A1-1, 2008.
- [3] 渡辺大, 岡本和人, 金子敏信, “軽量ハードウェア向け疑似乱数生成器 *Enocoro-128v2*,” 暗号と情報セキュリティシンポジウム, SCIS2010, 3D1-3, 2010.

A Sbox s_8

以下の表を用いて、8ビットの Sbox s_8 を実現することができる。

```
 $s_8[256] = \{$   
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  
 $\};$ 
```

B テストベクトル

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

...