疑似乱数生成器 MUGI

仕様書 Ver. 1.3

株式会社 日立製作所

2002 年 5 月 8 日

目次

1 序

本稿は、ストリーム暗号向けの疑似乱数生成器**MUGI** の仕様書である。MUGI は、秘密鍵 128 ビット、初期ベクトル (公開値)128 ビットをパラメータに持つ。

本稿では、まず 2 章で MUGI の設計指針を述べる。次に、3 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し、4 章で MUGI の仕様を詳述する。最後に、5 章で MUGI を実装する際に注意するべき点について触れる。

2 設計の指針

MUGI は、ソフトウェア、ハードウェアいずれの実装形態においても高速な (もしくは軽量な) 実装が可能であることを目標として設計されたストリーム暗号向けの疑似乱数生成器である。

ブロック暗号では、さまざまな実装形態で柔軟な実装が可能なアルゴリズムの設計手法が確立されているが、ストリーム暗号では、いずれかの実装形態に特化したアルゴリズムが一般的である。また、ソフトウェアに特化したアルゴリズムでは、安全性の評価が十分に行われていない場合も多く、パフォーマンスと安全性を両立する設計法は確立されていないのが現状である。

このような現状下で、我々は Panama[DC98] に注目する。Panama は 1998 年に Daemen と Clapp が提案した暗号モジュールであり、疑似乱数生成器、およびハッシュ関数として用いることができる。

Panama は、疑似乱数生成器の設計法として主流であった線形フィードバックシフトレジスタではなく、ブロック暗号と同じ原理に基づく設計を行っている。このため、ブロック暗号の設計、評価手法を適用しやすいと考えられる。また、基本的なアイデアが単純であり、同様の構造を持つバリエーションを設計しやすいことも特徴として挙げられる。一方で、Panama は従来にない設計を採用しており、Panama 自身の安全性評価は十分に行われているとは言い難い。

我々は、MUGI の設計方針として、Panama と同様の構造を持ち、安全性を十分に評価することを心掛けた。安全性に関する評価については、評価書 [Eval] を参照していただきたい。

結果として、MUGI は AES[FIPS-197] と同等程度の処理速度を達成可能であり、特にハードウェア上ではひじょうに高速である。また、安全性についても十分な評価を行ったと信じる。

以下、この章では PANAMA の構造の概略、すなわち MUGI の大域的な構造について 2.1 で、局所的な構造、部品の選択について 2.2 で簡単に述べる。

2.1 PANAMA 型鍵ストリーム生成器

一般に、疑似乱数生成器の主要部は内部状態 $\mathcal{S}$ とその状態遷移関数 $\mathcal{F}$、および内部状態から乱数列を出力する出力フィルター f の組 $(\mathcal{S}, \mathcal{F}, f)$ で構成される。特に、$(\mathcal{S}, \mathcal{F})$ の組を状態生成器 (internal-state machine) と呼ぶ。また、状態遷移 1 回を行うステップをラウンドと呼ぶ。ラウンド t における内部状態を明記するときには $\mathcal{S}^{(t)}$ と表す。

PANAMA の内部状態は 2 つの部分に別れる。それぞれをステート、バッファと称する。

次に、PANAMA の状態遷移関数は、ステート、バッファそれぞれの状態遷移関数に分離することができる (図 1)。ただし、各々の状態遷移関数はもう一方の内部状態をパラメータとして持つ。ステート、バッファの状態遷移関数をそれぞれ ρ 関数、λ 関数と称する。PANAMA の状態遷移関数の大きな特徴として、ρ 関数が SPN 構造を持つことが挙げられる。すなわち、ρ 関数はブロック暗号の段関数に相当する攪拌を行う。また、PANAMA の λ 関数は、簡単な線形変換 (フィードバックシフトレジスタ) である。

PANAMA の出力フィルターはラウンドごとにステートの約半分を乱数列として出力する。

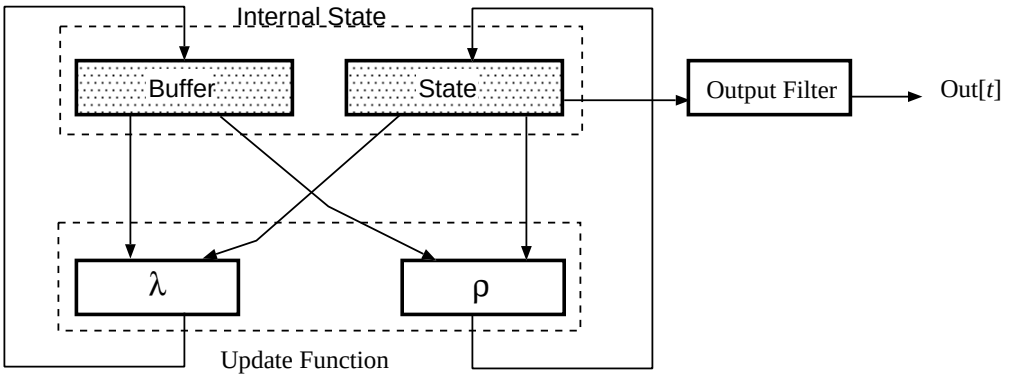

図 1: PANAMA 型鍵ストリーム生成器

以上の性質をまとめたものを以下、PANAMA 型鍵ストリーム生成器 (PANAMA-like keystream generator: PKSG) と呼ぶ。PKSG を定義としてまとめておく。

定義 1 2 つの部分ステート a、バッファ b を持つ内部状態と、それぞれの状態遷移関数 ρ 関数、λ 関数、出力フィルター f からなる疑似乱数生成器 $((a, b), (\rho, \lambda), f)$ が以下の条件

を満たすとき、PANAMA 型鍵ストリーム生成器という。

(1) ρ 関数は、a についての SPN 構造を含む非線形変換であり、バッファ b をパラメータに持つ。

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

(2) λ 関数は、b についての線形変換であり、ステート a をパラメータに持つ。

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

(3) 出力フィルター f はステート a の一部のみを出力する写像として定義される。すなわち、f はステート a の一部分 (1/2 以下) を乱数列として出力する。

2.2 部品の選択

MUGI を設計するに当たって、なるべく既存の良い技術を再利用することを目指した。結果として、MUGI では、評価が十分に行われている AES[FIPS-197] の構成要素 (例えば置換表 S-box) を多く用いている。この方針は、PKSG という比較的新しい技術を導入する上での安全性を高めると信じる。

3 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。

3.1 記号

$\oplus$	ビットごとの排他的論理和
$\wedge$	ビットごとの論理積
$\|\|$	二つのビット列の連結
$>>> n$	右まわりで n ビット巡回シフト (レジスタ幅は 64 ビット)
$<<< n$	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

3.2 データ構造

MUGI では、64 ビットのデータを基本とする。以下、64 ビットのデータブロックを

1 ユニットと呼ぶ。入力データ (秘密鍵、初期ベクトルなど) がバイト列で与えられた場合、ユニットへのデータ入出力は big endian で定義する。すなわち、8 バイトの入力情報 $x_0, \ldots, x_7$ が与えられたとき、1 ユニット a に対して次のように格納する。

$$a = [\mathrm{MSB}] \quad x_0||x_1||\ldots||x_7 \quad [\mathrm{LSB}].$$

ただし、[MSB] はユニットデータの最上位バイトを、[LSB] は最下位バイトを表す。またユニット内のバイトを参照する場合、x_i を “番号 i のバイト” と呼ぶ。データ出力は基本的にユニットデータとして出力する。

ユニット列内のバイト値を取り扱う場合には、1 番目の添字でユニットを、2 番目の添字でバイト番号を表すことにする。例えば、いくつかのユニット $b_0, \ldots, b_n$ からなるユニット列 $B = (b_i)_i$ に対し、$b_{i,j}$ はユニット b_i の、番号 j のバイトを表す。

また、ユニットデータ a の上位 32 ビットを a_H、下位 32 ビットを a_L と表す。

3.3 有限体上の演算

3.3.1 データの表現

本稿では、有限体 $\mathrm{GF}(2^8)$ 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、$\mathrm{GF}(2^8)$ の定義多項式 $\varphi(x)$ を

$$\varphi(x) = x^8 + x^4 + x^3 + x + 1 = \texttt{0x11b},$$

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

有限体の元は $\mathrm{GF}(2)$ 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、係数を降べキに並べた 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。

3.3.2 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、

$$\texttt{0x57} + \texttt{0xa3} = (x^6 + x^4 + x^2 + x + 1) + (x^7 + x^5 + x + 1)$$

$$= x^7 + x^6 + x^5 + x^4 + x^2$$
$$\leftrightarrow \texttt{0xf4}.$$

3.3.3 元の乗算

GF(2^8) 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(2^8) の元 $f(x) = \sum a_i x^i$ と x との乗算 $x \cdot f(x)$ を

$$\sum b_i x^{i+1} \bmod \varphi(x)$$

で定義する。たとえば、

$$\begin{aligned}\texttt{0x02} \cdot \texttt{0x87} &= x \cdot (x^7 + x^2 + x + 1) \\ &= x^8 + x^3 + x^2 + x \\ &= (x^4 + x^3 + x + 1) + x^3 + x^2 + x \\ &= x^4 + x^2 + 1 \\ &= \texttt{0x15}.\end{aligned}$$

$x^i \cdot f(x)$ は、上記定義から帰納的に計算できる。
GF(2^8) 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

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

で定義する。

3.3.4 逆元

$f, g \in$ GF(2^8) に対して、

$$f \cdot a + g \cdot b = 1 \bmod \varphi(x),$$

を満たす $a, b \in$ GF(2^8) が存在するとき、g は f の逆元と言い、$g = f^{-1}$ と表す。一般に有限体上の 0 を除く任意の元に対して逆元が存在することが知られている。GF(2^8) の場合には、

$$a^{-1} = a^{254},$$

で a の逆元が与えられる。

4 アルゴリズム

本章では、MUGI のアルゴリズムを記述する。2 章でも述べたように、疑似乱数生成器の主要部は状態生成器と出力フィルターの組合せとして記述される。本章では、まず 4.3 で MUGI の内部状態を、4.4 で状態遷移関数を書き下す。4.4 で詳細に触れなかった内容については、4.7 で触れる。また、4.5 で内部状態の初期化、4.6 で乱数の生成について述べる。

4.1 概要

MUGI は、128 ビットの秘密鍵 (秘密パラメータ)K、128 ビットの初期ベクトル (公開パラメータ)I、出力ユニット長 n(n は自然数) を入力として持ち、n ユニットの乱数列を出力する。アルゴリズムの概要は以下のとおりである。

入力: 秘密鍵 K、初期ベクトル I、出力ユニット長 n
出力: 乱数列 $Out[i]$ $(1 \leq i \leq n)$

アルゴリズム

初期化

Step 1. 鍵 K をステート部に入力し、ρ 関数を用いながらバッファ部を初期化する。

Step 2. 初期ベクトル I をステート部に入力し、ρ 関数を用いてステート部を初期化する。

Step 3. さらに内部状態全体の状態遷移を繰り返し、初期内部状態を形成する。

乱数生成

Step 4. n ラウンドの状態遷移を行う。ラウンドごとに内部状態の一部 (64 ビット) を出力する。

4.2 入力

MUGI は、入力として 128 ビットの秘密鍵 (秘密パラメータ)K、128 ビットの初期ベクトル (公開パラメータ)I を持つ。$K(I)$ の上位ユニット、下位ユニットをそれぞれ $K_0, K_1(I_0, I_1)$ と表す。

4.3 内部状態

4.3.1 ステート

ステート a は 3 ユニットで構成される。上位からそれぞれ a_0, a_1, a_2 と表す。

4.3.2 バッファ

バッファ b は 16 ユニットで構成される。上位から $b_0, \ldots, b_{15}$ と表す。

4.4 状態遷移関数

一般に、PKSG の状態遷移関数は、バッファをパラメータとしたステートの状態遷移関数 ρ 関数と、ステートをパラメータとしたバッファの状態遷移関数 λ の組み合わせとして記述される。すなわち、PKSG 全体の状態遷移関数を $Update$ と表すとき、

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

以下、MUGI の ρ 関数、λ 関数について説明する。

4.4.1 ρ 関数

ρ 関数はステート a の状態遷移関数で、2 つの F 関数を持つ拡張 Feistel 構造であり (図 2)、バッファ b をパラメータに持つ。ρ 関数は次の式で記述される。

$$\begin{aligned}
a_0^{(t+1)} &= a_1^{(t)}, \\
a_1^{(t+1)} &= a_2^{(t)} \oplus \mathrm{F}(a_1^{(t)}, b_4^{(t)}) \oplus C_1, \\
a_2^{(t+1)} &= a_0^{(t)} \oplus \mathrm{F}(a_1^{(t)}, b_{10}^{(t)} <<< 17) \oplus C_2.
\end{aligned}$$

上の式で、C_1, C_2 は定数である。F 関数は AES の構成要素 (S-box、行列変換) を利用している。F 関数の詳細については 4.7.3 で述べる。

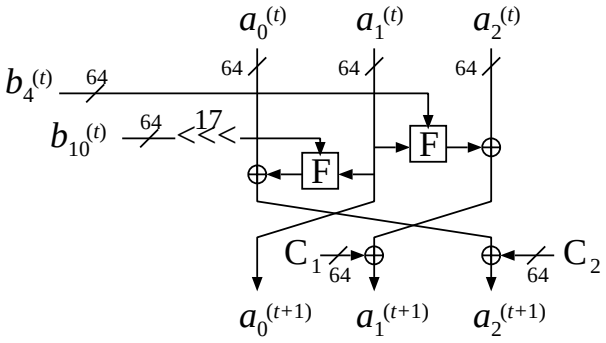

図 2: ρ 関数

4.4.2 λ 関数

λ 関数はバッファ b の状態遷移関数で、ステート a の状態をパラメータに持つ。λ 関数は線形写像であり、次の式で定義される。

$$
\begin{aligned}
b_j^{(t+1)} &= b_{j-1}^{(t)} \quad (j \neq 0, 4, 10), \\
b_0^{(t+1)} &= b_{15}^{(t)} \oplus a_0^{(t)}, \\
b_4^{(t+1)} &= b_3^{(t)} \oplus b_7^{(t)}, \\
b_{10}^{(t+1)} &= b_9^{(t)} \oplus (b_{13}^{(t)} <<< 32).
\end{aligned}
$$

4.5 初期化

MUGI の内部状態の初期化は、バッファ、ステートそれぞれ別に行い、さらに全体を攪拌する。

はじめに、鍵 K を用いてステートの初期化を行う。

$$
\begin{aligned}
a_0 &= K_0, \\
a_1 &= K_1, \\
a_2 &= (K_0 <<< 7) \oplus (K_1 >>> 7) \oplus C_0.
\end{aligned}
$$

C_0 は定数である。その後、ρ 関数のみによる状態遷移を行いながらバッファ b を初期化する。

$$b_{15-i} = (\rho^{i+1}(a, 0))_0.$$

ただし、ρ^i は ρ 関数の i 回繰り返しを、$\rho(a, 0)$ はバッファ b からの入力が 0 であることを意味する。すなわち、ステート部に鍵 K をセットした後、ρ 関数でステートの内部状態

を遷移し、その上位ブロックをバッファに順次セットする。ここで、バッファ内部に格納されたデータはステート部にはフィードバックしないものとする。

次に、バッファの初期化終了後のステート a の状態 $a(K) = \rho^{16}(a_0, 0)$ と初期ベクトル I を用いて、ステート a の内部状態を形成する。まず、

$$\begin{aligned} a(K,I)_0 &= a(K)_0 \oplus I_0, \\ a(K,I)_1 &= a(K)_1 \oplus I_1, \\ a(K,I)_2 &= a(K)_2 \oplus (I_0 <<< 7) \oplus (I_1 >>> 7) \oplus C_0, \end{aligned}$$

としてステート a を再初期化し、16 ラウンドの ρ 関数のみによる攪拌を行う。この時点でのステートの状態は $\rho^{16}(a(K,I),0)$ と表すことができる。さらに、16 ラウンドの状態遷移による内部状態全体の攪拌を行う。

すなわち、

$$a^{(1)} = Update^{16}(\rho^{16}(a(K,I),0), b(K)).$$

ただし、$b(K)$ は鍵 K による初期化を完了したバッファの状態を表す。

4.6 乱数の生成

MUGI は、内部状態の初期化後、状態遷移を繰り返しながら各ラウンドごとに 64 ビットの乱数を生成する。ラウンド t における出力を $Out[t]$ とすると、この出力は

$$Out[t] = a_2^{(t)},$$

で与えられる。すなわち、MUGI はラウンド処理のはじめにステートの下位 64 ビットを出力する。

初期化から乱数生成までの処理は表 1 に従う。

表 1: 乱数生成のスケジュール

	ラウンド t	処理内容	入力	出力
初期化	-49	鍵の入力	K	–
	$-48, \ldots, -33$	内部状態の撹拌 (ρ 関数)	–	–
	-32	初期ベクトルの入力	I	–
	$-31, \ldots, -16$	内部状態の撹拌 (ρ 関数)	–	–
	$-15, \ldots, 0$	内部状態の撹拌 ($Update$)	–	–
乱数生成	$1, \ldots$	乱数出力、内部状態の撹拌	–	$Out[t]$

4.7 部品

この節では、4.4、4.5 で説明無しで用いたいくつかの項目について詳細に記述する。特に、4.4.1 の F 関数は、本疑似乱数生成器の主要部である。F 関数は、ブロック暗号で良く用いられる SPN 構造を持ち、バイト単位の置換表 S-box と、GF(2^8) 上の 4×4 行列で記述される線形変換を利用する。

以下、4.7.1 で S-box について、4.7.2 で行列変換について、4.7.3 で F 関数全体の構成について、4.7.4 で MUGI で使用する定数について説明する。

4.7.1 S-box

8 ビットの置換表である S-box には AES に使用されたものを用いる。すなわち、S-box の与える置換は有限体 GF(2^8) 上の逆元操作 $x \rightarrow x^{-1}$ とアフィン変換の合成で定義される (ここでは、$0^{-1} = 0$ とする、またビット位置を示す添え字の大きいものが上位ビットである)。

$$
\begin{aligned}
b' = S(x) \Leftrightarrow & \\
& b = x^{-1}, \\
& \begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}.
\end{aligned}
$$

この S-box をテーブル参照として実現したものを、付録 A に示す。

4.7.2 行列

F 関数の線形変換層は、バイト単位の 4×4 行列による線形変換と、バイト単位の置換から構成される。MUGI では、この行列として、S-box と同じく AES で採用された MDS 行列を用いる。この行列を M と表せば、4 バイトデータ $X = x_0||x_1||x_2||x_3$ の行

列 M による変換は次で記述される:

$$M(x) = M \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} \texttt{0x02} & \texttt{0x03} & \texttt{0x01} & \texttt{0x01} \\ \texttt{0x01} & \texttt{0x02} & \texttt{0x03} & \texttt{0x01} \\ \texttt{0x01} & \texttt{0x01} & \texttt{0x02} & \texttt{0x03} \\ \texttt{0x03} & \texttt{0x01} & \texttt{0x01} & \texttt{0x02} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix}.$$

バイト単位の演算は 3.3 で定義したものを用いる。

上記行列変換で用いられる乗算の乗算数は、0x01, 0x02, 0x03 のみである。このうち、乗算数が 0x01 の場合は恒等変換であり、$\texttt{0x03} = \texttt{0x01} \oplus \texttt{0x02}$ と表現できるので、本質的に乗算として実装する必要があるのは、乗算数が 0x02 の場合のみである。また、ソフトウェア実装する場合には、これらの乗算数に対応するテーブルを用意しておくことで、乗算処理を高速に行うことができる。乗算数が0x02 の場合の乗算表は付録 B を参照のこと。

4.7.3 F 関数

F 関数は、鍵加算 (バッファのデータ加算)、S-box を用いた非線形変換、線形変換 (MDS 行列 M による行列変換+バイト置換) の合成で与えられる (図 3)。F 関数への 64 ビットの入力を (X, B) とし、出力を Y とすると、F 関数による変換は次の式で記述される:

$$\begin{aligned} Y = F(X, B) \Leftrightarrow \\ & O = X \oplus B, \\ & O_0||O_1||O_2||O_3||O_4||O_5||O_6||O_7 = O, \\ & P_i = S(O_i) \quad (0 \le i < 8), \\ & P_H = P_0||P_1||P_2||P_3, \quad P_L = P_4||P_5||P_6||P_7, \\ & Q_H = M(P_H), \quad Q_L = M(P_L), \\ & Q_0||Q_1||Q_2||Q_3 = Q_H, \quad Q_4||Q_5||Q_6||Q_7 = Q_L, \\ & Y = Q_4||Q_5||Q_2||Q_3||Q_0||Q_1||Q_6||Q_7. \end{aligned}$$

S-box と行列 M は、ソフトウェア実装で処理する場合、同時に処理することができる [FIPS-197]。この高速化手法により、F 関数は複雑な処理を高速に行うことができる。

4.7.4 定数

MUGI で使う定数は、初期化で用いられる C_0、ρ 関数で用いられる C_1, C_2 がある。こ

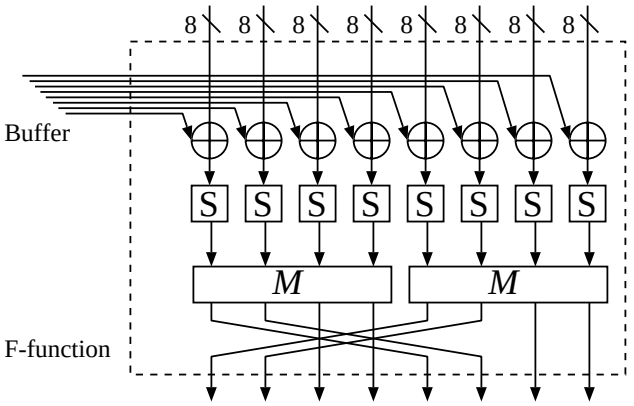

図 3: F 関数

れらの定数は、次で与える:

$$\begin{aligned} C_0 &= \texttt{0x6A09E667F3BCC908}, \\ C_1 &= \texttt{0xBB67AE8584CAA73B}, \\ C_2 &= \texttt{0x3C6EF372FE94F82B}. \end{aligned}$$

これらの定数は、それぞれ $\sqrt{2}, \sqrt{3}, \sqrt{5}$ を 2^{64} 倍した数の整数部分の下位 64 ビットを 16 進表示したものである。

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。

5.2 暗号化と復号化

MUGI 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 64 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 (4)。復号化もまったく同様にして実現できる。

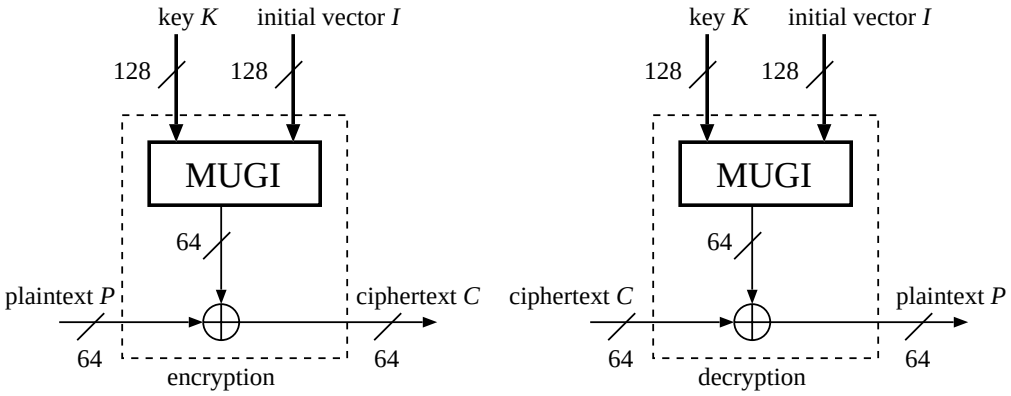

図 4: MUGI を用いた暗号化と復号化

参考文献

[DC98] J. Daemen, C. Clapp, "Fast Hashing and Stream Encryption with PANAMA," *Fast Software Encryption, 5th International Workshop, FSE'98, Proceedings*, LNCS Vol. 1372, Springer-Verlag, 1998.

[FIPS-197] National Institute of Standards and Technology, Federal Information Processing Standards Publication 197, Advanced Encryption Standard (AES).

[WFT01] 渡辺 大, 古屋聡一, 宝木和夫, "F 関数を利用した鍵ストリーム生成器の設計法," 暗号と情報セキュリティシンポジウム, SCIS 2001-6A-4, 2001.

[WFST01a] 渡辺 大, 古屋聡一, 瀬戸洋一, 宝木和夫, "ソフトウェアに適した擬似乱数生成器の提案," 電子情報通信学会技術研究報告, ISEC2001-8, 2001.

[WFST01b] 渡辺 大, 古屋聡一, 瀬戸洋一, 宝木和夫, "PANAMA 型擬似乱数生成器の乱数性," 電子情報通信学会技術研究報告, ISEC2001-57, 2001.

[Eval] 渡辺大, 古屋聡一, 吉田博隆, 宝木和夫, 疑似乱数生成器 MUGI 自己評価書, 2001, available at `http://www.sdl.hitachi.co.jp/crypto/mugi/index-j.html`.

A S-box

以下の表を用いて、S-box 置換を実現することができる。

$$S(x) = \mathrm{Sbox}[x]$$

Sbox[256] = {

```
0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5,
0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0,
0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc,
0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a,
0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0,
0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b,
0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85,
0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5,
0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17,
0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88,
0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c,
0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9,
0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6,
0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e,
0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94,
0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68,
0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16  };
```

B 乗算数が0x02の場合の乗算表

以下の表を用いて、乗算数が0x02の場合の乗算を実現することができる。

$$\texttt{0x02} \cdot x = \mathrm{mul2}[x]$$

mul2[256] = {

```
0x00, 0x02, 0x04, 0x06, 0x08, 0x0a, 0x0c, 0x0e,
0x10, 0x12, 0x14, 0x16, 0x18, 0x1a, 0x1c, 0x1e,
0x20, 0x22, 0x24, 0x26, 0x28, 0x2a, 0x2c, 0x2e,
0x30, 0x32, 0x34, 0x36, 0x38, 0x3a, 0x3c, 0x3e,
0x40, 0x42, 0x44, 0x46, 0x48, 0x4a, 0x4c, 0x4e,
0x50, 0x52, 0x54, 0x56, 0x58, 0x5a, 0x5c, 0x5e,
0x60, 0x62, 0x64, 0x66, 0x68, 0x6a, 0x6c, 0x6e,
0x70, 0x72, 0x74, 0x76, 0x78, 0x7a, 0x7c, 0x7e,
0x80, 0x82, 0x84, 0x86, 0x88, 0x8a, 0x8c, 0x8e,
0x90, 0x92, 0x94, 0x96, 0x98, 0x9a, 0x9c, 0x9e,
0xa0, 0xa2, 0xa4, 0xa6, 0xa8, 0xaa, 0xac, 0xae,
0xb0, 0xb2, 0xb4, 0xb6, 0xb8, 0xba, 0xbc, 0xbe,
0xc0, 0xc2, 0xc4, 0xc6, 0xc8, 0xca, 0xcc, 0xce,
0xd0, 0xd2, 0xd4, 0xd6, 0xd8, 0xda, 0xdc, 0xde,
0xe0, 0xe2, 0xe4, 0xe6, 0xe8, 0xea, 0xec, 0xee,
0xf0, 0xf2, 0xf4, 0xf6, 0xf8, 0xfa, 0xfc, 0xfe,
0x1b, 0x19, 0x1f, 0x1d, 0x13, 0x11, 0x17, 0x15,
0x0b, 0x09, 0x0f, 0x0d, 0x03, 0x01, 0x07, 0x05,
0x3b, 0x39, 0x3f, 0x3d, 0x33, 0x31, 0x37, 0x35,
0x2b, 0x29, 0x2f, 0x2d, 0x23, 0x21, 0x27, 0x25,
0x5b, 0x59, 0x5f, 0x5d, 0x53, 0x51, 0x57, 0x55,
0x4b, 0x49, 0x4f, 0x4d, 0x43, 0x41, 0x47, 0x45,
0x7b, 0x79, 0x7f, 0x7d, 0x73, 0x71, 0x77, 0x75,
0x6b, 0x69, 0x6f, 0x6d, 0x63, 0x61, 0x67, 0x65,
0x9b, 0x99, 0x9f, 0x9d, 0x93, 0x91, 0x97, 0x95,
0x8b, 0x89, 0x8f, 0x8d, 0x83, 0x81, 0x87, 0x85,
0xbb, 0xb9, 0xbf, 0xbd, 0xb3, 0xb1, 0xb7, 0xb5,
0xab, 0xa9, 0xaf, 0xad, 0xa3, 0xa1, 0xa7, 0xa5,
0xdb, 0xd9, 0xdf, 0xdd, 0xd3, 0xd1, 0xd7, 0xd5,
0xcb, 0xc9, 0xcf, 0xcd, 0xc3, 0xc1, 0xc7, 0xc5,
0xfb, 0xf9, 0xff, 0xfd, 0xf3, 0xf1, 0xf7, 0xf5,
0xeb, 0xe9, 0xef, 0xed, 0xe3, 0xe1, 0xe7, 0xe5   };
```

C テストベクトル

key[16] = {0}
iv[16] = {0}
output =
  0xc76e14e70836e6b6, 0xcb0e9c5a0bf03e1e,
  0x0acf9af49ebe6d67, 0xd5726e374b1397ac,
  0xdac3838528c1e592, 0x8a132730ef2bb752,
  0xbd6229599f6d9ac2, 0x7c04760502f1e182,
  ...
key[16] =
{0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f}
iv[16] =
{0xf0, 0xe0, 0xd0, 0xc0, 0xb0, 0xa0, 0x90, 0x80,
0x70, 0x60, 0x50, 0x40, 0x30, 0x20, 0x10, 0x00}
output =
  0xbc62430614b79b71, 0x71a66681c35542de,
  0x7aba5b4fb80e82d7, 0x0b96982890b6e143,
  0x4930b5d033157f46, 0xb96ed8499a282645,
  0xdbeb1ef16d329b15, 0x34a9192c4ddcf34e,
  ...