疑似乱数生成器 MUGI

自己評価書

株式会社 日立製作所

2001 年 9 月 25 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 の自己評価書である。

一般に、暗号プリミティブにおいては、十分な評価が行われた後に、はじめて安全である (良い暗号である) と認められる。本評価書の目的は、設計者による安全性および実装に関する評価の結果を開示することである。これは、MUGI に興味を持った人が、自分で評価を行う際の助けになると信じる。また、開発者が慎重に開発を行ったという主張でもある。

本稿では、まず、2 章で本稿の議論に必要な記号、用語について述べる。MUGI のアルゴリズムに関しては、仕様書 [Spec] を参照していただきたい。次に、3 章で MUGI の暗号モジュールとしての安全性について、4 章で MUGI のソフトウェア実装とハードウェア実装について論じる。

以上の評価内容を持って、我々は MUGI が「十分に良い」疑似乱数生成器であると信じる。

2 準備

この章では、本稿で用いる記号及び略号を記述する。

2.1 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 進表記を表す接頭語

2.2 略号

LFSR	線形フィードバックシフトレジスタ (Linear Feedback Shift Register)
PRNG	疑似乱数生成器 (Pseudorandom Number Generator)
PKSG	Panama 型鍵ストリーム生成器 (Panama-like Keystream Generator)

PKSG の定義は仕様書 [Spec] を参照のこと。

3 安全性評価

この章では、MUGI の安全性について検討した結果について述べる。MUGI は、Panama 型鍵ストリーム生成器 (PKSG) の一種である。PKSG とは、1998 年に Daemen と Clapp が提案した暗号モジュール Panama の乱数生成モードを一般化したものである [WFT01] [WFST01a]。PKSG の定義は仕様書 [Spec] を参照のこと。

PKSG は、ブロック暗号の段関数と線形フィードバックシフトレジスタ (LFSR) の組み合わせと見ることもできるが、従来とはまったく異なる設計を採用している。このため、PKSG に対する有効な評価手法は確立されていないのが現状である。

この章では、MUGI の安全性について、いくつかの異なる面から検討していく。一般に、疑似乱数生成器に求められる要件は以下の 2 つである。

(1) 出力列が十分な乱数性を持つこと
(2) 異なる初期値を与えた場合に出力列が大きく変化すること

本章では、まず要件 (1) について、我々は MUGI の出力列に対していくつかの数値テストを行った。この結果について 3.1 で述べる。次に、3.2 でそれ以外の従来の理論的な乱数列評価手法の適用の可否について簡単に触れる。3.3 では、既存の方法ではなく、PKSG の乱数性評価手法として最も効率的であると我々が考えている手法を紹介し、この手法を MUGI に適用した結果について述べる。

(2) に関しては、評価内容をさらに 2 つに分けることができる。一つは、秘密鍵を固定し、初期ベクトルのみを変化させた場合の出力列の変動に関するものである。この評価に関する一般的な議論は、[DGV94] で行われている。この評価に関する結果は 3.4 で述べる。

もう一つは、初期ベクトルを固定し、秘密鍵を変化させた場合の出力列の変動に関するものである。これは、鍵関連攻撃の一種と考えることができる。この評価に関する結果に

ついては 3.5 で述べる。また、3.5 では、秘密鍵と初期ベクトルの間に見られる簡単な相関についても触れる。

3.1 乱数性に関する数値テスト

本節では、MUGI の出力に対して、いくつかの乱数性をチェックするテストを行った結果について述べる。暗号学的安全性が要求される擬似乱数列の乱数性をチェックする方法は、FIPS140-1([FIPS]) に以下の 3 つが記載されている。

(1) 1 ビットの頻度検定
(2) 4 ビットの頻度検定
(3) 連の検定 (長すぎる連の検出を含む)

しかし、[FIPS] に記載されているテストでは、テストする擬似乱数列の長さが短いため、ストリーム暗号に用いる擬似乱数列の安全性を保証することはできない。我々は、FIPS 140-1 に準拠した乱数性テストに比べ、十分に長い平文長に対して頻度検定を行った。

3.1.1 頻度テスト

ここでは、1 ビット、2 ビット、4 ビット、8 ビットに対して検定を行った。検定の方法は [Kn81] にもとづく。平文長の違いによる乱数性の変化を観察するために、平文長が $2^{22}, 2^{26}, 2^{30}$ ビットの場合に対してテストを行った。鍵、攪拌用パラメータは、Rijndael を乱数生成関数として用いることにより 512 組生成して使用した。実際には、C 言語標準ライブラリの乱数生成関数により生成させた乱数を Rijndael に入力し、その出力を、鍵、攪拌用パラメータとして使用した。
次に、テストの結果について述べる。それぞれのテスト結果を表 1、表 2、表 3、表 4 にまとめた。表の値は、それぞれのテストを、ある乱数列の長さに対して行い、ある棄却率で検定した場合に、乱数ではないと判定された初期値の組の数である。例えば、2^{30} ビットの乱数列に対して、1 ビットの頻度検定を行った結果を見ると、512 個中 5 個の初期値が出力する乱数列が 99% の確率で真の乱数ではない、と言える。

いずれのテストについても、初期値の組が、真の乱数と区別できる乱数列を出力する割合は、棄却率にほぼ等しい。以上のテストの結果、MUGI の出力は真の乱数と区別できるとは言えない。

表 1: 乱数列テストの結果（1 ビットの頻度検定）

	棄却された鍵の個数 (/512)	
乱数列の長さ (bit)	棄却率 0.05	棄却率 0.01
2^{22}	25	6
2^{26}	20	4
2^{30}	22	5

表 2: 乱数列テストの結果（2 ビットの頻度検定）

	棄却された鍵の個数 (/512)	
乱数列の長さ (bit)	棄却率 0.05	棄却率 0.01
2^{22}	19	3
2^{26}	23	8
2^{30}	18	3

表 3: 乱数列テストの結果（4 ビットの頻度検定）

	棄却された鍵の個数 (/512)	
乱数列の長さ (bit)	棄却率 0.05	棄却率 0.01
2^{22}	18	6
2^{26}	31	6
2^{30}	16	3

表 4: 乱数列テストの結果（8 ビットの頻度検定）

	棄却された鍵の個数 (/512)	
乱数列の長さ (bit)	棄却率 0.05	棄却率 0.01
2^{22}	17	6
2^{26}	18	6
2^{30}	18	1

3.1.2 連テスト

ここでは、検定の方法は [MOV97] にもとづいてテストを行った。鍵、攪拌用パラメータの生成方法は、3.1.1 と同様である。
次に、テストの結果について述べる。表の値は、テストを、乱数列の長さが 2^{22} ビットの場合に対して行い、ある棄却率で検定した場合に、乱数ではないと判定された初期値の組の数である。例えば、512 個中 3 個の初期値が出力する乱数列が 99% の確率で真の乱数ではない、と言える。

表 5: 乱数列テストの結果（連検定）

	棄却された鍵の個数 (/512)	
乱数列の長さ (bit)	棄却率 0.05	棄却率 0.01
2^{22}	29	3

表 5 から、このテストについても、初期値の組が、真の乱数と区別できる乱数列を出力する割合は、棄却率にほぼ等しい。また、同様の条件で長い連の検出も行った結果、最も長い連の長さは 31 で、その個数は 1 個であった。乱数列の長さが 2^{22} ビットのとき、長さ 31 の連が存在する頻度の期待値は 2^{-11} である。512 個の鍵で試行を行ったので、4 回に 1 回は長さ 31 の連の検出が見込まれる。従って、このテストの結果は妥当なものである。以上のテストの結果、MUGI の出力は真の乱数と区別できるとは言えない。

3.2 既知の乱数性評価手法の適用検討

3.2.1 周期

疑似乱数生成器の出力する列は、鍵長に見合う長い周期を持つべきである。MUGI の場合には、鍵長が 128 ビットなので、任意の秘密鍵 K、初期ベクトル I が生成する乱数列は周期 2^{128} 以上を持つことが望ましい。

MUGI は、状態遷移関数が非線形であるため、その出力列の周期を評価することは困難である。しかし、内部状態を保持する空間のサイズ、および状態遷移関数の構成から、ブロック長 128 ビットのブロック暗号の OFB モードと同等以上の長い周期を持つと期待できる。

3.2.2 線形複雑度

線形複雑度は、状態遷移関数が LFSR を用いている場合に、特に有効となる乱数性評価手法であり [Ru86]、MUGI に対して、Berlekamp-Massey のアルゴリズムを用いた攻撃法を適用することは困難と考える。

3.2.3 分割統治攻撃 (divide-and-conquer attack)

分割統治攻撃に分類される攻撃法は、内部状態の一部を推定する攻撃であり、あるラウンドにおける推定から、任意のラウンドの内部状態 (一部) が記述できる場合に適用可能である。

しかし、PKSG では一般に内部状態が大きく、また、状態遷移関数を分割して、内部状態の一部のみを記述することができない。このような理由から、MUGI に対して分割統治攻撃を適用することは困難であると考える。

ただし、分割統治攻撃は再同期攻撃と組み合わせて適用することも考えられる。再同期攻撃に対する安全性の評価に関しては、3.4 を参照のこと。

3.3 乱数性の評価

一般的なビット列の乱数性を検証する方法として、さまざまなものが知られている。これらのほとんどは、直接的にビット列の統計的な情報を収集する方法である。この他にも、LFSR に基づいた PRNG の出力列に対して、理論的に乱数性を評価する手法が多く知られている。しかし、前節でも述べたように、これらの手法で PKSG の出力列が十分な (鍵長に見合う) 乱数性を持つかどうかを検証することは困難である。

我々は、PKSG がブロック暗号に類似した構造を持つことに注目し ([WFT01] [WFST01a])、ブロック暗号の安全性評価手法、特に線形解読法 [Ma93] を用いて、出力列の distinguisher を構成する、すなわち、出力列と真の乱数列を区別することを試みる。

3.3.1 F 関数の差分・線形特性

MUGI の F 関数は、鍵加算、8 ビットの置換表 S-box による非線形変換、4×4 の MDS(最大距離分離符号) 行列 M による線形変換、バイト位置の入れ替えからなる SPN 構造を持つ (図 1)。

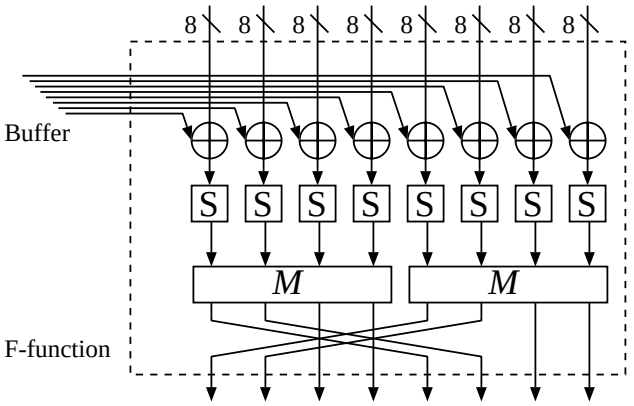


図 1: MUGI の F 関数

この S-box、および行列 M は AES[DR99] で用いられているものと同じ変換である。すなわち、S-box の最大差分・線形確率はいずれも 2^{-6} である。ただし、線形確率は正規化した確率である。また、行列 M が定義する線形変換の分岐数は 5 である。

したがって、2 つの F 関数を近似する差分パス

$$\Delta_0 \xrightarrow{\mathrm{F}} \Delta_1 \xrightarrow{\mathrm{F}} \Delta_2$$

を考えると、このパスの最大差分特性確率は 2^{-30} となる。また、上記パスにおいて $\Delta_0 = \Delta_2$ となる場合には、さらに active S-box の数が多くなる。これはバイト置換による効果である。この場合の active S-box の最小個数は 10 であり、最大差分特性確率は 2^{-60} となる。

線形パスの場合も同様であり、

$$\Gamma_0 \xrightarrow{\mathrm{F}} \Gamma_1 \xrightarrow{\mathrm{F}} \Gamma_2$$

で与えられる線形パスの最大線形特性確率は 2^{-30}、$\Gamma_0 = \Gamma_2$ となる場合の最大線形特性確率は 2^{-60} となる。

3.3.2 線形解読法の適用

線形解読法は、ブロック暗号の入出力の特定ビットに注目し、これらのビットの排他的論理和が偏差を持つ場合に秘密鍵の推定を行う攻撃法である。これは、入出力の linear correlation を評価することに他ならない。

我々は、線形解読法のステップのうち、上記の偏差を観測する技術 (すなわち distinguisher の構成法) に注目する。我々の提案する評価手法は、PKSG がラウンドごとに出

力するユニットの特定ビットの和 (マスク値、ラウンドごとに異なってよい) を計算し、その結果として出力列のみからなる線形近似式を構成する。もし、高い確率で成り立つ線形近似式が存在すれば、PKSG の出力する列は真の乱数では無いと言える。PKSG にこの手法を適用する場合、ブロック暗号の場合と異なり、バッファの値 (ブロック暗号では、段鍵に相当する。[WFT01] 参照) は動的に更新されており、このことが線形近似式の構成を困難にしている。我々は、実際に線形近似式を構成するのではなく、線形近似式を与えるパスが満たすべき条件から、可能な線形パスの active S-box の下限を計算することを試みる。

結論として、本評価手法では、MUGI の出力列と真の乱数を区別することはできなかった。

定理 1 MUGI の出力列からなる線形近似式は、少なくとも 22 個の active S-box を持つ。

以下、この結果に証明を与えていく。ただし、ある時刻において、内部状態は十分に攪拌されているものとする。上で述べたように、線形解読法を PKSG の出力する乱数列に適用するには、出力列のみからなる近似式を構成する必要がある。我々は、これを以下の 2 つのステップに分けて検討した。

1. ρ 関数の近似式の構成
2. バッファを含めた線形パスの探索

ここで、線形パスとは線形近似式を与えるマスクデータの流れを意味する。以下、それぞれのステップについて説明する。

ρ 関数の線形近似

まず、評価を容易にするために ρ 関数の等価変形を行う (図 2)。図中では、アルゴリズムの等価変形を見やすくするため、F 関数の一方を G と表記している。また、定数 C_1、C_2 の排他的論理和は無視した。以下の議論で ρ 関数という場合には、等価変更を行った後の関数を意味するものとする。

このとき、1 段もしくは複数段の線形パスは、以下の 2 つの近似式

$$\Gamma_1 \cdot a_0^{(t)} \oplus \Gamma_2 \cdot a_2^{(t)} = \Gamma_2 \cdot a_0^{(t+1)}, \tag{1}$$

$$\Gamma_1 \cdot a_0^{(t)} \oplus \Gamma_1 \cdot a_2^{(t+1)} = \Gamma_2 \cdot a_0^{(t+1)}, \tag{2}$$

の組み合わせで与えられる (付録 A 図 9)。ただし、上の式で $\Gamma \cdot x$ はマスク Γ とデータ x の内積を表す。図 9 中、太線は active なパスを表す。式 (1), (2) は、それぞれ F 関数、G 関数 1 個のみを近似するパスである。

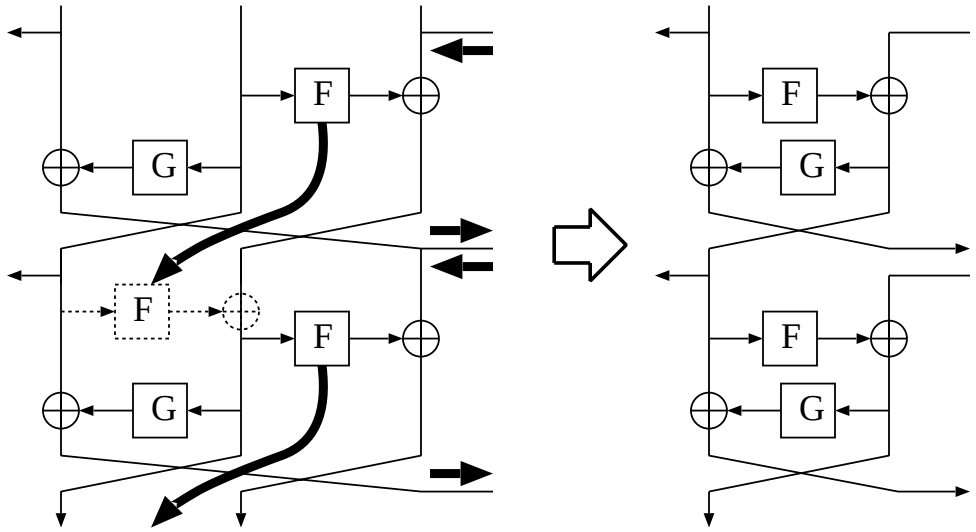


図 2: 線形パスの等価変形

上記パスを組み合わせて得られる線形パスのうち、以下の議論で重要なものを付録 A 図 10 に挙げておく。局所的に active S-box の数が 5 以上であることが保証できるのは図 10 に挙げた 5 パターンのみである。MUGI のアルゴリズムでは、ブロック暗号とは異なり、線形拡散層の分岐数から直ちに active S-box の数が保証できるわけではないことに注目されたい。

すべての線形パスに共通の条件

次に、バッファを含めた線形パスの探索を行う。PKSG では、ブロック暗号の場合と異なり、パスを構成する段数 (ラウンド数) が固定されていない。このため、計算機を使ってすべてのパスを探索することは不可能である。

そこで、我々は、パスの始まるラウンド t_s(以下、始点) とパスの閉じるラウンド t_e(以下、終点)、およびその周辺に求められる条件から、理論的に各パスの active S-box の個数の下限を計算した。以下、各線形パスの active S-box の下限を $\mathcal{AS}$ と表す。MUGI では、S-box の線形特性確率が 2^{-6} なので、$\mathcal{AS} < 22$ となる線形パスが存在しないならば、出力列の linear correlation は十分小さいといえる。

また、ステートの線形パスをバッファを考慮しながら結合させる場合に重要となるのが、ステートからバッファへの排他的論理和されるデータのマスクである。以下、このマスクを特に $\Gamma(D)^{(t)}$ と表すことにする。

まず、始点と終点のラウンドについて考える。始点では、バッファ、ステートすべてのユニットの入力マスクは 0 であり、出力ユニット $\mathrm{Out}[t_s]$ のみが active となる。ここで、あるユニットが active とは、ユニットのマスクが 0 マスクでは無いことを表す。この条

件を満たすステートの時刻 t_s におけるマスクパターンは、図 10 の Case 1, 3 のみである。同様にして、終点の条件を満たす時刻 t_e のマスクパターンは Case1, 2 のみとなる。

次に、バッファからの影響について検討する。始点では、バッファのすべての入力マスクが 0 マスクとなるので、$\Gamma(D)^{(t)}$ は t_s+4 ラウンドまで 0 マスクとなる。また、t_s において、バッファの G 関数への入力は active となるので、$\Gamma(D)^{(t_s+5)}$ は active となる。同様の議論から、$\Gamma(D)^{(t)}$ は $t_e-5 \le t \le t_e$ で 0 マスク、時刻 t_e-6 では active となる。

active S-box の個数の下限の評価

表記を簡単にするために、ラウンド t において F 関数、G 関数が active、もしくは 0 近似であることを、それぞれ 1, 0 で表すことにする。例えば、ラウンド t で F 関数が active であり、G 関数を 0 近似する場合には、$\Gamma(a)^{(t)} = (1,0)$ と表記する。

パスの探索 ($\mathcal{AS}$ の下限の計算) はいくつかの場合に分けて行う。ラウンド t_e-6〜t_e-1 において、次の条件が成り立つ場合を考える。

条件T1: $(\Gamma(a)^{(t_e-i)}, \Gamma(a)^{(t_e-i+1)}) = ((0,0),(1,1))$ を満たすラウンド $i(2 \le i \le 7)$ が存在する。

条件T2: $1 \le i \le 7$ のすべての i に対して、$\Gamma(a)^{(t_e-i)} \neq (0,0)$ が成り立つ。

始点、終点における ρ 関数のパスに関する考察と同様にして、条件T1 とT2 は互いに相補的であることがわかる。

ラウンド t_s+1〜t_s+4 においても、

$$(\Gamma(a)^{(t_s)}, \Gamma(a)^{(t_s+1)}) = ((1,1),(0,0))$$

の場合に

条件H1: $1 \le i \le 4$ のすべての i に対して、$\Gamma(a)^{(t_s+i)} = (0,0)$ が成り立つ。

条件H2: H1 の相補条件

を考える。

以上の準備の下に、次の 4 つの場合それぞれについて、active S-box の個数の下限を計算する。

I. 始点: Case 1, 終点: Case 1
II. 始点: Case 1, 終点: Case 2
III. 始点: Case 3, 終点: Case 1
IV. 始点: Case 3, 終点: Case 2

(1) I の場合

この場合、始点と終点が Case 1 なので、少なくとも $\mathcal{AS} \geq 20$ であることが保証される。また、$\Gamma(D)^{(t_e-11)}$ が active となる。

さらに、線形パスが条件H2 を満たす場合には、Case 1 もしくは Case 3 のパスをもう一つ含むことになるので、$\mathcal{AS} \geq 25$ となる。

一方、線形パスが条件H1 を満たすとすれば、$\Gamma(a)^{t_s+5} \neq (0,0)$ が成り立つ。また、$\Gamma(D)^{(t_s+6)}, \Gamma(D)^{(t_e-6)}$ が active、$\Gamma(D)^{(t_e-7)}$ は 0 近似となる。したがって、$t_e - t_s \geq 14$ でなければならない。さらに、$\Gamma(D)^{(t_e-6)}$ が active であることから、$\Gamma(a)^{(t_e-6)} \neq (0,0)$ もしくは $\Gamma(a)^{(t_e-7)} \neq (0,0)$ が成立する。すなわち、この場合には少なくとも $\mathcal{AS} \geq 22$ となる。

(2) II の場合

この場合、始点と終点で $\mathcal{AS} \geq 15$ が保証される。

まず、条件T1 が成り立つ場合について考える。このとき、線形パスは Case 3 を 1 つ含むので、$\mathcal{AS} \geq 20$ となる。$\Gamma(a)^{(t_e-6)} \neq (0,0)$ ならば明らかに $\mathcal{AS} \geq 25$ である。また、Case 2 となるラウンドが $t_e - 4$ 以前であれば、$\mathcal{AS} \geq 22$ となる。したがって、$\Gamma(a)^{(t_e-2)} = 0$ としてよい。以上の条件下では、$\Gamma(D)^{(t_e-10)}$ は active となるので、I の後半の議論と同様にして $\mathcal{AS} \geq 22$ であることがわかる。

次に、条件T2 が成り立つ場合を考える。このとき、$\mathcal{AS} \geq 21$ である。さらに、終点付近のパスが Case 4 もしくは 5 を含めば、$\mathcal{AS} \geq 24$ となる。そうでない場合には、ラウンド $t_e - 12$〜$t_e - 7$ のうち、2 ラウンドに 1 つ $\Gamma(D)^{(t)}$ が active になる。したがって、$\mathcal{AS} \geq 24$ であることがわかる。

(3) III の場合

この場合も、始点と終点で $\mathcal{AS} \geq 15$ を保証する。また、ラウンド $t_s + 1$〜$t_s + 4$ のパスは、active S-box が少なくとも 4 以上であることを保証するので、$\mathcal{AS} \geq 19$ である。

さらに、終点付近のパスについては、今までの議論と同様にして、

$$\Gamma(a)^{(t_e-i)} = (0,0), \quad 1 \leq i \leq 6$$

としてよい。このとき、

$$\Gamma(D)^{(t_e-i)} = 0, \quad 7 \leq i \leq 9$$

となる。したがって、ラウンド $t_e - 10$〜$t_e - 7$ のパスは、4 つ以上の active S-box を持つ。したがって、この場合には $\mathcal{AS} \geq 23$ となる。

(4) IV の場合

この場合、少なくとも $\mathcal{AS} \geq 19$ である。条件T2 が成り立つ場合には、III の場合と同様の議論で $\mathcal{AS} \geq 22$ であることがわかる。

条件T1が成り立つ場合、Case 3 となるラウンドを $t_e - i_0$ とすると、ラウンド $t_e - i_0 + 1$ 〜$t_e - 1$ のパスは、少なくとも $i_0 - 1$ の acitve S-box を持つから、$\mathcal{AS} \geq 19 + (i_0 - 1)$ となることがわかる。したがって、II の場合の議論と同様にして、$i_0 \leq 3$ としてよい。このとき、まず、$\Gamma(D)^{(t_e - 10)}$ が active となる。すなわち、ラウンド $t_s + 5$ と $t_e - 7$ は一致しない。したがって、ラウンド $t_e - 7$ の active S-box (必ず存在する) は、上記計算では未カウントである。この結果、$i_0 \leq 2$ としてよい。以下、同様の議論を繰り返すことにより、すべての場合について $\mathcal{AS} \leq 22$ であることが検証できる。

3.3.3 その他の攻撃法

ブロック暗号に対するその他の攻撃法、例えば、差分解読法 [BS93]、高階差分攻撃 [Ku94]、補間攻撃 [JK97] などは、いずれも選択平文攻撃である。これらの攻撃法を (乱数性の評価手法として) PKSG に適用することは困難であると考えられる。これは、ブロック暗号の場合と異なり、攻撃者は任意の出力列を得ることができないからである。したがって、乱数列に対する攻撃は既知平文攻撃のみを考えればよい。

また、任意の選択平文攻撃において、出力列から攻撃に必要な選択平文を取り出すことは計算量的に困難である。例えば、16 ラウンドの出力列から何らかの distinguisher を構成できた場合、出力列の成す空間の元の数は $2^{64\times 16}$ であり、求める選択平文 1 組を得るためには $2^{64\times 8}$ 程度の既知平文が必要である。

さらに、差分解読法を例にとって考える。差分解読法で、3.3.2 と同様に差分パスを探索する場合、まず、攻撃者はあるラウンドにおける内部状態の差分を自由に観測できる必要がある。これは、初期化が十分に行われている場合には不可能であり、したがって、差分解読法の適用は困難であると考えられる。

3.4 MUGI に対する再同期攻撃

この節では、再同期攻撃 (re-syncronization attack)[DGV94] の MUGI への適用を検討する。

まず、再同期攻撃について簡単に説明する。再同期攻撃とは、秘密鍵の他に公開パラメータを持つ鍵ストリーム生成器に対して適用可能な攻撃法であり、特に、初期化が十分に行われていない場合に有効である。再同期攻撃では、鍵を固定した状態で、公開パラメータと出力する乱数列の相関を調べ、そこから秘密鍵を導出する攻撃法一般を指す。たとえば、ブロック暗号のカウンターモードに対する線形解読法は再同期攻撃の一例で

ある。

我々は、入出力の相関として、特に差分・線形特性に注目する。差分解読法 [BS93]、線形解読法 [Ma93] はブロック暗号の強力な評価手法として知られている。一方、PKSG はステートの状態遷移関数 ρ としてブロック暗号の段関数と同様の構造を採用している。したがって、ρ 関数を繰り返してデータ攪拌を行った場合に、内部状態と初期値の相関を調べる方法として、上記 2 つの攻撃法は効果的と考えられる。

3.4.1 データ出力を伴わない ρ 関数の差分・線形パス

まず、ステート a からデータ出力を伴わない場合、すなわち、バッファ b への排他的論理和および乱数の出力を含まない状態での ρ 関数の差分・線形パスについて検討する。これは、通常のブロック暗号の差分・線形パスの探索とまったく同様である。

評価の方法は、それぞれの攻撃法に対して、active な F 関数の個数を総当たりで数える手法を採った。MUGI の S-box の最大差分・線形確率は 2^{-6}、線形拡散層の最小分岐数は 5 である (3.3.1)。MUGI の F 関数は 1 層の SPN 構造なので、active な (入力差分もしくは入力マスクが 0 マスクではない) F 関数の数から直接 active S-box の数の最小値を保証することはできないが、ラウンド t における各ユニット $a_i^{(t)}$ に対する active な F 関数の数が 10 以上ならば、入力データとステート $a^{(t)}$ の相関は十分に小さいと考えられる。

表 6: ρ 関数の差分・線形パスにおける active な F 関数の数

Number of rounds	$\cdots$	11	12	13	14	15	16	17	18	19	20	21
Differential	$\cdots$	4	5	6	6	6	7	8	8	8	9	10
Linear	$\cdots$	4	5	6	6	6	7	8	8	8	9	10

それぞれの攻撃法に対して、各ユニットに影響を与える (active な) F 関数の個数の最小値を表 6 に示す。ただし、入力はステート a のすべての状態を取り得るものとして計算を行った。

3.4.2 再同期攻撃に対する安全性

表 6 は、ステート a のすべてのユニットに対して、初期ベクトル I と ρ 関数で t 回変換したステート $a^{(t)}$ との相関を表したものである。表 6 によれば、$t = 21$ で最大差分特性確率、最大線形特性確率ともに飽和すると考えられる。

MUGI の初期化では、初期ベクトル I を入力してから、16 ラウンドの ρ 関数のみによるステート a の攪拌を行う。この後、状態遷移関数 $Update$ による 16 ラウンドの攪拌が行われるが、差分パス、線形パスのいずれも、バッファ b の存在がステート a のパスに影響を与えるのは-9 ラウンド目以降、すなわち、初期ベクトル I を入力してから 22 ラウンド目以降である。したがって、$t > 0$ 以降では、ステート内部は初期ベクトルとの相関は十分に小さいと考えられる。

3.4.3 再同期攻撃と分割統治攻撃の併用

表 6 より、ラウンド 0(初期化終了直後) において、初期ベクトル I とバッファ b の一部の相関は十分に小さいとは言えない。しかし、出力列とバッファからなる差分の関係式、もしくは線形近似式は 2 ユニット以上のバッファ値を含む。このバッファ値のいずれかは、初期ベクトル I との相関が十分に小さい。したがって、攻撃者は上記相関を観測することはできない。

3.5 鍵セットアップの設計と解析

鍵スケジュール部は、出力される乱数列がじゅうぶんにランダムとなるようにバッファとステートの初期化を行うアルゴリズムであり、その目的で設計された。

本稿では、鍵のセットアップについて 3 点の側面から議論し、各々独立な章で議論してゆく。

3.5.1 線形バッファの（不）可能性

一般にバッファの値は、秘密鍵に対して非線形な関数により初期化される。これはバッファの初期化に、非線形な ρ 関数を用いているからである。しかしながら、ある特別な場合には、非線形性が落ちることも考えられる。このうちの一つの場合として、いつくもの ρ 関数の入力が同じとなる場合である。この場合、入力、すなわち秘密鍵、と出力、すなわち初期化されたバッファに線形表現が成り立つ場合がある。ここでは、バッファに線形な関係が起こるような場合について考察する。

バッファの初期化は、二つの段階がある。一つは鍵によるバッファの初期化 (A) であり、もう一つは鍵、初期値の両方がセットされたあとのバッファの攪拌 (B) である。まず、一つの考察として次のことが考えられる; 「(A) が十分非線形度をもつような鍵の場合、最終的に（乱数を出力する直前の）バッファが線形な関係を持つことは起こりにくい

(ただし、未知変数を 16 ワード以上はとらない、と仮定)」。この考察から、(A) において十分多くの鍵が複雑な非線形度をもって攪拌されることを検証し、またそうでないと考えられる鍵の個数がひじょうに小さいことを示す。

これらのひじょうに限定される鍵についても、さらに IV を伴った非線形な攪拌がバッファ全体に渡って行われることから、結果として、鍵セットアップでは、バッファを十分ランダムに攪拌すると結論付けられる。

■Case 1: (1 **段繰り返し**)

定理 2 ρ 関数には固定点がない。すなわち、すべてのバッファに同じ値を初期化する鍵はない。

Proof

A_0, A_1, A_2 と B_0, B_1, B_2 をそれぞれ、ρ 関数の入力、出力とする。ρ 関数へのバッファからの入力はここでは 0 であるので、ρ 関数の定義より、A と B の関係は以下のようになる。

$$B_0 = A_1, \tag{3}$$
$$B_1 = A_2 \oplus F(A_1, 0) \oplus C_1, \tag{4}$$
$$B_2 = A_0 \oplus F(A_1, 0) \oplus C_2. \tag{5}$$

ここで (A_0, A_1, A_2) を固定点とする。このとき、以下の関係が成り立つ。

$$B_i = A_i, i = 0, 1, 2. \tag{6}$$

式 (3)(4) (5)(6) より、A が固定点となるための条件が以下のようになる;

$$C_1 = C_2.$$

C_1, C_2 の定義より、これは成り立たない。よって ρ 関数は固定点を持たない。

■Case 2: (2 **段繰り返し表現**)

定理 3 ρ 関数を 2 回繰り返す構造では、たくさんの固定点は存在しない。よって、2 通りの値のみでバッファを初期化する鍵の明らかなクラスは存在しない。

Proof

2 段 ρ 関数はそれ自身でたくさんの固定点を持つ。しかし、128 ビットの鍵を 192 ビットのバッファへセットする場合に適用するパディングがこれらのほとんどを不可能な入力とする。

よって、ここの証明ではパディングを考慮する。a_0 と a_1 を秘密鍵のそれぞれ上位、下位とする。このときパディングである a_2 は以下のように定義される:

$$(a_0 <<< 7) \oplus (a_1 >>> 7) \oplus C_0.$$

ここで a_2 の生成について線形性と他のワードのセットとの独立性について注目する。繰り返し表現を考える場合には、鍵のセットのために以下の鍵のセットを考えても一般性を失わない:

$$\begin{aligned} a_0 &= ((a_1 >>> 7) \oplus C_0 \oplus a_2) >>> 7, \\ a_1 &= \text{秘密鍵上位}, \\ a_2 &= \text{秘密鍵下位}. \end{aligned}$$

$\alpha = F(a_1, 0)$、$\beta = F(a_2 \oplus \alpha \oplus C_1, 0)$ とする。2 段 ρ 関数の出力は以下のようになる:

$$(a_2 \oplus \alpha \oplus C_1, a_0 \oplus \alpha \oplus C_2 \oplus \beta \oplus C_1, a_1 \oplus \beta \oplus C_2).$$

よって、(a_0, a_1, a_2) が 2 段 ρ 関数の固定点となるための必要条件は以下のようになる:

$$\begin{aligned} a_1 &= \text{任意}, \\ a_2 &= F^{-1}(\beta, 0) \oplus C_1 oplus F(a_1, 0), \\ a_0 &= a_1 \oplus C_1 \oplus \beta \oplus C_2 \oplus \alpha \\ &= a_2 \oplus C_1 \oplus \alpha. \end{aligned}$$

ある固定した a_1 について (2^{64} とおり)、上記条件を満たす a_2 は平均ひとつ存在する。これは a_2 が以下の式で定義されることから明らか:

$$a_2 = f^{-1}(a_2 \oplus C_a, 0) \oplus C_b,$$

ここで、C_a と C_b は a_1 依存の定数。a_0 は a_1 と a_2 から一意に決定される。よって、パディングを考えない場合、(a_0, a_1, a_2) の組には約 2^{64} 個の 2 段 ρ 関数の固定点がある。ここでの見積もりは荒いが、パディングによりほとんどの (a_0, a_1, a_2) が生成不能となる。よって、このような弱鍵のクラスはとても小さく、意味のある攻撃を考えることはできない。

■Case 3: (3 **段繰り返し表現**)

定理 4 ρ 関数を 3 回繰り返す構造では、固定点は存在しない。

Proof

ここではパディングの性質を用いない。(a_0, a_1, a_2) を秘密鍵により初期セットされたステートとする。このとき 3 段処理の出力 $(a_0^{(3)}, a_1^{(3)}, a_1^{(3)})$ は以下のようになる:

$$\begin{aligned} a_0^{(3)} &= a_0 \oplus \alpha \oplus \beta \oplus C_1 \oplus C_2, \\ a_1^{(3)} &= a_1 \oplus \beta \oplus \gamma \oplus C_1 \oplus C_2, \\ a_2^{(3)} &= a_2 \oplus \gamma \oplus \alpha \oplus C_1 \oplus C_2, \\ \alpha &= F(a_1, 0), \\ \beta &= F(a_2 \oplus \alpha \oplus C_1, 0), \\ \gamma &= F(a_0 \oplus \alpha \oplus \beta \oplus C_1 \oplus C_2, 0). \end{aligned}$$

ここでもし (a_0, a_1, a_2) が 3 段の繰り返し表現となる場合、

$$(a_0, a_1, a_2) = (a_0^{(3)}, a_1^{(3)}, a_1^{(3)})$$

となる。これらより、繰り返し表現となるための必要十分条件として以下の式が導かれる。

$$\alpha \oplus \beta = C_1 \oplus C_2, \tag{7}$$
$$\beta \oplus \gamma = C_1 \oplus C_2, \tag{8}$$
$$\gamma \oplus \alpha = C_1 \oplus C_2. \tag{9}$$

式 (7), (8) より、

$$\gamma \oplus \alpha = 0. \tag{10}$$

となる。この式 (10) は、残りの条件 (式 (9)) と矛盾する。よって、3 段繰り返し表現となる入力は存在しない。

■Case 4: **(4 段以上の繰り返し表現について)** 我々は 4 段以上の場合についても考慮した。しかし、以下の理由から、どの場合についても、「弱鍵として指摘できるほどの弱点を持たない」と結論づける。

適用可能な秘密鍵の数: 繰り返し出力を生成するための必要条件は通常、二つの複雑なワード長の方程式からなる。これは 192 ビットのステートの値のうち平均 2^{64} 個がこの性質を持つと考えることができる。*Case 2* と同様に、パディングにより、このような鍵による弱鍵のクラスはなくなると考える。

繰り返しバッファの効果: 繰り返しの値によるバッファの初期化の効果は、後に続くバッファ-ステートを含めたデータの撹拌により、さらに弱められる。さらに、バッファには少なくとも 4 つの値が存在するため、これにより、初期化に現れる単純な性質は発生しにくいと考えられる。

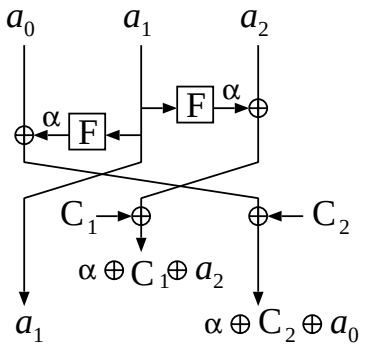


図 3: 1 段繰り返し

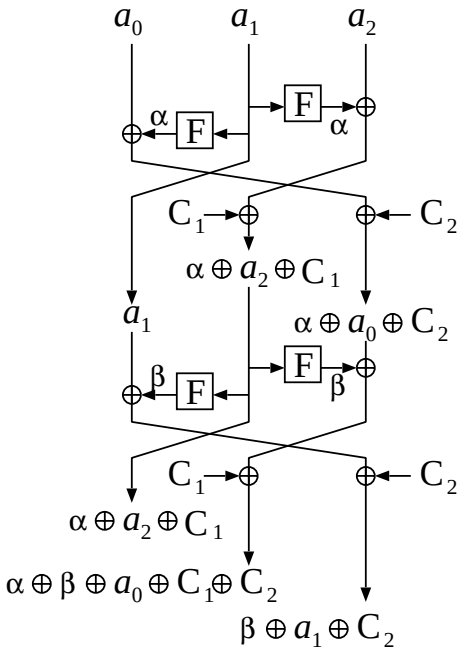


図 4: 2 段繰り返し

3.5.2 Square 攻撃

MUGI は byte-oriented な構造であるため、SQUARE 攻撃 [DKR97] の適用を検討する必要がある。SQUARE 攻撃は Rijndael などの SPN 構造のブロック暗号に対してひじょうに強力な手法である。ここでは、この攻撃の適用可能性について検討し、可能であるような性質についての解析をまとめる。結果として、SQUARE 攻撃のどのような変形につい

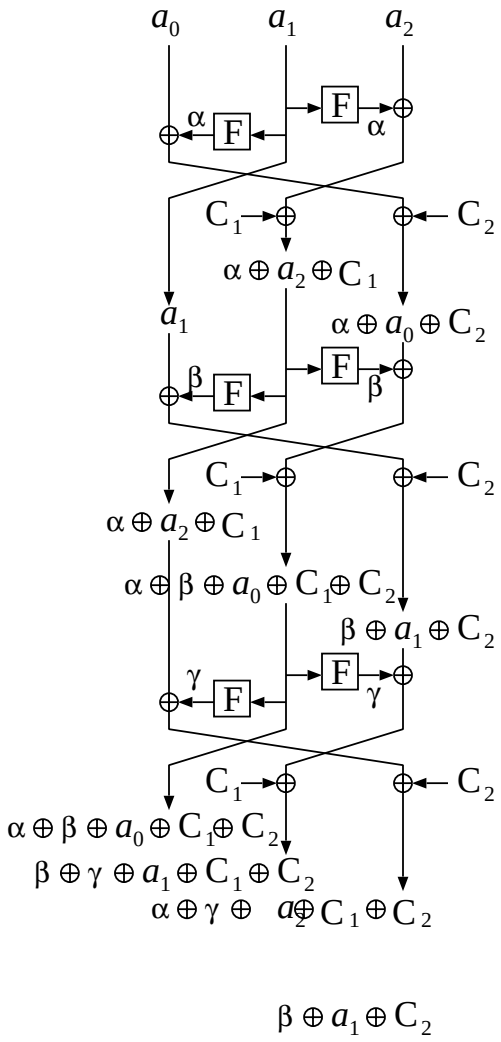


図 5: 3 段繰り返し

ても、フルスペックの MUGI に対する安全性に影響を及ぼさないと結論付ける。

ブロック暗号に対する SQUARE 攻撃は、基本的に選択平文攻撃である。攻撃者は、典型的には 1 バイト (またはワード) だけ異なるような関連を持つ平文を多数集める。非線形関数への入力が飽和するため、攻撃者はある程度中間値を制御できる。暗号文側からは攻撃者は鍵をゲスしながら、飽和した平文により制御できている中間値を部分的に復号する。攻撃者が部分的復号化の際、鍵をゲスしているならば、攻撃者は正しい段鍵と誤りのものを区別することができる。

ストリーム暗号では、攻撃者は鍵か初期値のどちらかに差分を入れなければならない。よってストリーム暗号への SQUARE 攻撃の適用には、関連鍵攻撃か、選択初期値攻撃の

表 7: 各々の中間値におけるワードの性質

Intermediate value	Word property
$(a_0^{(0)}, a_1^{(0)}, a_2^{(0)})$	(Λ, O, O)
$(a_0^{(1)}, a_1^{(1)}, a_2^{(1)})$	(O, O, Λ)
$(a_0^{(2)}, a_1^{(2)}, a_2^{(2)})$	(O, Λ, O)
$(a_0^{(3)}, a_1^{(3)}, a_2^{(3)})$	$(\Lambda, \Lambda, \Lambda)$
$(a_0^{(4)}, a_1^{(4)}, a_2^{(4)})$	(Λ, Φ, Φ)
$(a_0^{(5)}, a_1^{(5)}, a_2^{(5)})$	$(\Phi, *, *)$
$(a_0^{(6+)}, a_1^{(6+)}, a_2^{(6+)})$	$(*, *, *)$

みが考えうる。

■関連鍵攻撃: まず、攻撃のモデルを定義する。攻撃者は鍵の値は知らないものとする。「飽和状態」を生成するために、攻撃者は複数回の鍵初期化を実行する (一度の実行を run と呼ぶ)。各々の run では、一部分の鍵が異なっている。特にここでは、word 毎の鍵差分による、run を考える。攻撃者は疑似乱数生成が始まるまで内部状態について何も知ることができない。ここでは、攻撃者がこれら複数の run 間に現れる、出力列の性質に着目する。

飽和した鍵のグループは、この飽和状態がバッファ初期化に反映される。まず、どのようにこの飽和状態が初期化に係わってくるかを解析する。簡単なため、鍵のパディングについてはここでは言及しない。これにより攻撃者はステートの初期状態として、最大の柔軟性を持って攻撃を考えることができる。Λ を「異なる run で常に異なる値である」ような中間ワードの性質、すなわち飽和状態、とする。全ての run についてワードが定数であるような性質を O とする。また最も弱い性質である、“balance” な状態 Φ を定義する。これは、すべての run の値を xor した結果が 0 である状態である。もし、あるワードがこれらのどの状態でもない、すなわち制御不能、な場合、これを $*$ と表す。もし、ワードの三組 (A, B, C) について、A, B, C がそれぞれ Λ, O, and Φ の状態であるとき、$(A, B, C) \xrightarrow{p} (\Lambda, O, \Phi)$、もしくは $A \xrightarrow{p} \Lambda, B \xrightarrow{p} O$, かつ $C \xrightarrow{p} \Phi$ と表記する。

明きからに最も効果的な飽和の埋め込みは、他のワードになるべく遅く影響するワードへの埋め込みを行うことである。そこで、$(a_0, a_1, a_2) \xrightarrow{p} (\Lambda, O, O)$ の場合について解析する。t 段の処理の出力を $(a_0^{(t)}, a_1^{(t)}, a_2^{(t)})$ と表記するので、これらの結果は表 (7) のようになる。

よって、バッファ b_i の初期値はインデックス i に応じて、以下の性質を持つ：

$$b_i \xrightarrow{p} \begin{cases} O & : \quad i = 15, 14, \\ \Lambda & : \quad i = 13, 12, \\ \Phi & : \quad i = 11, \\ * & : \quad i = 10, 9, ..., 0 \end{cases} \tag{11}$$

ここで、b_{11} までしか制御できないわけではないことに注意する。実際のところ b_{11} は、他のバッファの和と一度の F 関数の適用で表記できる (これについては非線形バッファの関係の項参照) からである。しかしながら、IV のセットのあとに続く撹拌により、この性質は出力列が生成するまでには崩されてしまう。よって、SQUARE 攻撃に基づく関連鍵攻撃は適用不能と考える。

■選択初期値攻撃: この攻撃は上記の選択鍵攻撃よりは現実的である。しかし、IV は 14 段の撹拌を終えるまでは、バッファへの影響がない。上で示した制御可能な段数を考慮すると、16 段の撹拌により、IV による飽和性質は十分に破壊される。

3.5.3 非線形バッファ相関

初期バッファは秘密鍵のみから生成される。鍵セットアップアルゴリズムは、1 段につき各々の初期ユニットを初期化する。鍵セットアップの段関数は、ランダム置換ではないため、初期バッファ値感に関連がある。ここではバッファ値間の関係について解析する。

ここで考える段関数はバッファからのフィードバックがないため、二つの F 関数は同じ入力となる。よって ρ 関数の処理は軽くなる一方、線形な関係はより簡単になる。解析の結果、いくつかのバッファユニットは一度の F 関数の評価と他のユニットの線形和で表すことができることがわかった。ここでは、バッファ間の関係の簡単な例のいくつかを表に示す。

$$\begin{aligned} b_{15} &= a_1, \\ b_{14} &= a_2 \oplus F(b_{15}, 0) \oplus C_1, \\ b_{13} &= a_0 \oplus a_2 \oplus b_{14} \oplus F(B_{14}, 0) \oplus C_2, \\ b_i &= a_0 \oplus a_1 \oplus a_2 \oplus b_{i+1} \oplus b_{i+2} \oplus F(b_{i+1}, 0) \oplus C_2, \\ & \qquad i = 12, 11, 10, \ldots, 0. \end{aligned}$$

3.5.4 等価バイト性質

ここでは Panama 型鍵ストリーム生成器に関する興味ある性質について議論する。F 関数の構造により以下の性質が述べられる。

■性質：F 関数における等価バイト性質 S ボックス層への入力のバイトがすべて同じ場合、対応する出力も同じ性質をもつ。すなわち出力の全部のバイトが同じ結果となる。

Proof

性質 Ω_n を n ビットレジスタに関する「すべてのバイトが同じ値」であることとする。S ボックス層の入力が Ω_{32} はであるとき、明らかに出力もそうである。さらに、一般に MDS が Ω の性質を持つとは限らないが、MUGI や Rijndael で用いられている MDS には Ω_{32} の性質がある。(x_1, x_2, x_3, x_4) と (y_1, y_2, y_3, y_4) をそれぞれ MDS の入力、出力とする。このとき、

$$
\begin{aligned}
y_1 &= \texttt{0x02}x_1 \oplus \texttt{0x03}x_2 \oplus \texttt{0x01}x_3 \oplus \texttt{0x01}x_4, \\
y_2 &= \texttt{0x01}x_1 \oplus \texttt{0x02}x_2 \oplus \texttt{0x03}x_3 \oplus \texttt{0x01}x_4, \\
y_3 &= \texttt{0x01}x_1 \oplus \texttt{0x01}x_2 \oplus \texttt{0x02}x_3 \oplus \texttt{0x03}x_4, \\
y_4 &= \texttt{0x03}x_1 \oplus \texttt{0x01}x_2 \oplus \texttt{0x01}x_3 \oplus \texttt{0x02}x_4.
\end{aligned}
$$

(x_1, x_2, x_3, x_4) が Ω_{32} である、すなわち $x_1 = x_2 = x_3 = x_4$ のとき、$y_1 = y_2 = y_3 = y_4 = x_1$ となる。ゆえに、MDS の出力も Ω_{32} となる。この性質から、F 関数への性質へと発展させる。F 関数には S ボックスと MDS からなるデータの流れが二つある。よって、性質 Ω を崩さないためには、64 ビットの S ボックス層への入力が Ω_{64} であることが必要となる。

この F 関数の性質により、より外側の構造への議論へある程度発展させる。ここで F 関数の両方の入力が Ω_{64} であるとき、出力も同じ性質を持っていたことを思い出す。よって、バッファ、ステートすべての 64 ビットレジスタが Ω_{64} であるとき (ただし、レジスタ間の値は異なってもよい)、非線形変換 F はこの性質を変化させない。しかしながら、後に続く定数の排他的論理和により、この性質は崩壊する。よって、この定数の XOR により、性質 Ω に基づく攻撃は MUGI の攻撃へは効果的ではないと考える。

4 実装評価

MUGI はソフトウェア、ハードウェアのいずれのプラットフォーム上でも高速かつ軽量な実装が可能となるような設計を行っている。この章では、MUGI のソフトウェア実

装、ハードウェア実装についてそれぞれ述べる。

4.1 ソフトウェア実装

MUGI は、ρ 関数に AES の構造を採り入れている。また、ρ 関数以外は、排他的論理和、巡回シフトなどの簡単な論理演算だけで構成されている。このような構成の特性から、MUGI は任意のプラットフォーム上で高速に動作すると考えられる。

本節では、汎用プロセッサである Intel®社の Pentium®III 上で MUGI を実装した結果について述べる。実装は C 言語で行った。評価環境の詳細は表 8 を参照のこと。

表 8: ソフトウェア評価環境

ハードウェア	CPU	Pentium®III 800MHz
	RAM	512Mbyte
ソフトウェア	OS	Microsoft®Windows®2000
	コンパイラ	Microsoft®Visual C++ 6.0
	言語	ANSI C
	最適化オプション	速度最適化

MUGI を表 8 の環境で実装した場合、実装コストは表 9 のとおり。

表 9: メモリ使用量

コード量		619 行
ワークエリア	初期化	4.6 Kbyte
	乱数生成部	4.2 Kbyte

ここで、コード量はソースファイルから空白行、コメント行を取り除いた行数として測定した。また、ワークエリアの測定では乱数列を格納するメモリは、ワークエリアから除外した。

この環境において、MUGI は 294Mbps の処理を行うことができる。この結果を clock/byte に換算したものが表 10 である。この速度は状態遷移関数を空回しして測定したものである。すなわち、メモリへのデータ書き込みなどに要する時間は無視している。また、初期化には約 15000clock を要する。

上記の速度は参考数値であり、十分に高速化されたプログラムによるものとは言い難い。また、MUGI では 64 ビットのデータを基本ブロックサイズ (ユニット) としている。

表 10: ソフトウェア処理速度

初期化	15029 clock
乱数生成	21.8 clock/byte

このため、64 ビットプロセッサ上では、さらに高速な処理が期待される。

4.2 ハードウェア実装

本節では、MUGI のハードウェア評価の結果について述べる。我々は、MUGI のハードウェア実装を行うに当たり、次の二つの方式を検討した。

(1) 速度優先方式
(2) 小論理規模方式

以下に、検討と実装 (論理合成) の結果を述べる。実装評価は、合成ツール Synopsys Design Compiler、セルライブラリ Hitachi ASIC HG73C(0.35 μ m CMOS プロセス) を用いて行った。評価の結果、速度優先方式では、論理規模 26Kgate 程度で処理速度 2.9Gbps を得た。

4.2.1 速度優先方式

速度優先方式では、高速化のために処理をできるだけ並列化することを考える。そこで、MUGI の並列化の検討を行う。MUGI は、PKSG の一種であり、内部状態と状態遷移関数を持ち、内部状態を元にして乱数が生成される。内部状態は状態遷移関数 F により遷移し次の内部状態となる。このため、時刻 $t+1$ の内部状態を求めるため時刻 t の内部状態が必要である。したがって、複数の状態遷移関数を並列に実装しても 1 つの乱数系列に関して高速化は見込めない。

そこで、速度優先方式では、MUGI の 1 つの状態遷移関数の構成要素をすべて実装し、可能な限り構成要素が並列に動作するようにする。

図 6 に MUGI ハードウェアの全体構成図を示す。MUGI ハードウェアは鍵レジスタ K、初期ベクトルレジスタ I、ステートレジスタ a、バッファレジスタ b、鍵入力処理ブロック init1、初期ベクトル入力処理ブロック init2、ρ 関数ブロック、λ 関数ブロック、および制御ブロック control から構成される。

control が外部からの信号を受け、各処理ブロックを制御する。各処理ブロックは並列

に動作し、1 クロックで 1 ラウンドの乱数を出力する。

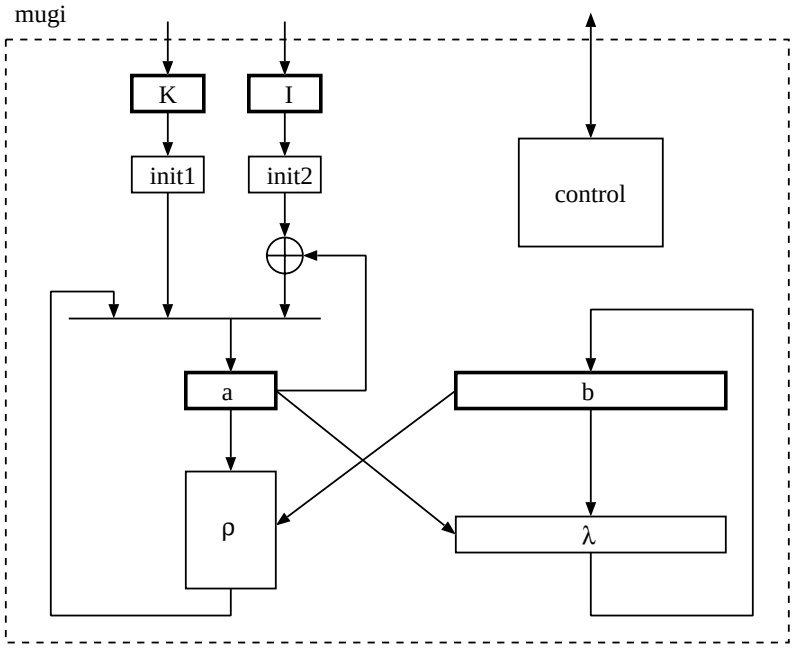


図 6: MUGI ハードウェア全体構成図

図 7 に、ρ 関数ブロックの構成図を示す。速度優先方式の ρ 関数は 2 つの F 関数ブロックと 64 ビット排他的論理和で構成される。各 F 関数は 8 つの S-box ブロック (S) と 2 つの線形変換層ブロック (MDS) と 8 つの 8 ビット排他的論理和で構成される。これにより ρ 関数ブロックは 1 クロックで 1 ユニットの乱数生成に必要なデータを処理することができる。

上記のような方式で実装した結果を表 11 に示す。

表 11 中の数字は、各モジュールのゲート数を示す。また、「mugi(全体)」は mugi を構成するモジュール全体をまとめて合成したものである。そのため、最適化により個別のモジュールのゲート量の合計よりも小さくなっている。合成時の最適化は論理を最小化するように行った。

この実装方式で実装した場合、スループットは動作周波数 45.7MHz で 2922Mbps となる。ただし、スループットは「mugi(全体)」のものである。また、初期化には 1095ns を要する。

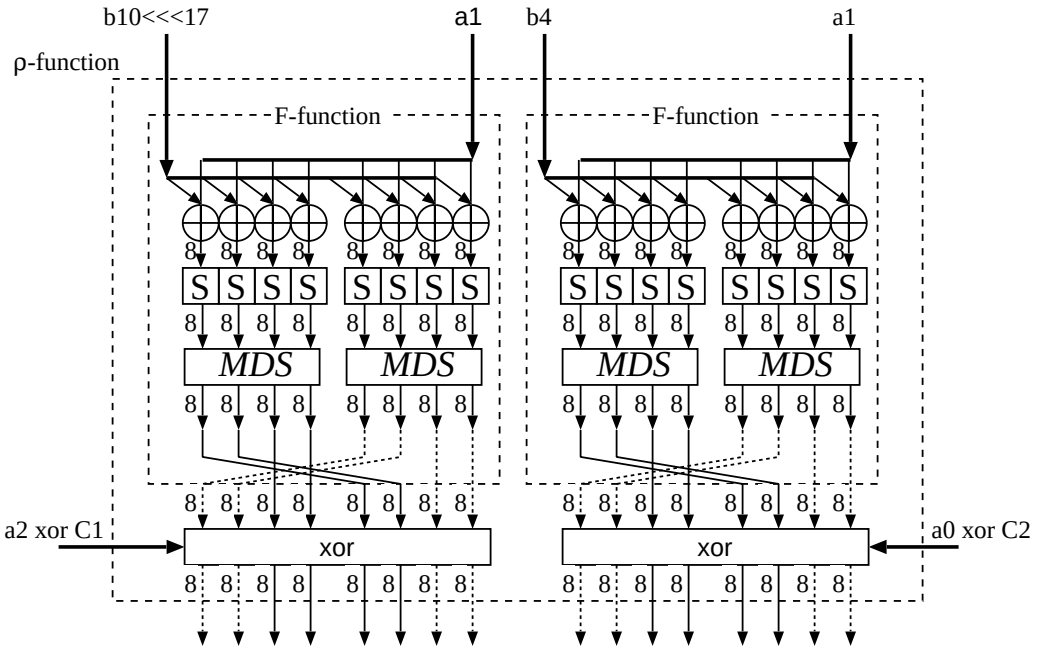


図 7: ρ 関数ブロック構成図 (速度優先)

表 11: ゲート規模 (速度優先方式)

モジュール名	下位モジュールを含むゲート数	モジュール単体のゲート数
mugi(全体)	**26068**	**26068**
mugi	28918	14034
control	133	133
ρ	12562	360
F 関数	6101	128
MDS	307	307
S-box	670	670
init	181	181
λ	1826	1826

4.2.2 小論理規模方式

小論理規模方式では、ゲート量を削減するために、同じ機能のブロックをできるだけ統合することを考える。そこで、速度優先方式での実装を統合の出発点とすると、図 6 の MUGI ハードウェア全体構成図では、鍵入力処理ブロック init1 と初期ベクトル入力処理ブロック init2 の機能がほぼ等しいため統合することが可能である。これにより、64 ビットの排他的論理和が二つ削減できる。しかし、これらを統合した場合、統合したブロックの入力に鍵レジスタ K と初期ベクトルレジスタ I のための 128 ビットセレクタが付くことになる。これらの増減を考慮すると、init1 と init2 の統合は不要であると考えられる。

次に、図 7 の ρ 関数ブロック構成図を見ると、S-box ブロックが 16 個、線形変換層ブロックが 4 個ある。そこで、これらのブロックを統合し論理規模の削減を考え、4 つの線形変換層ブロックを 1 個、16 個の S-box ブロックを 4 個に統合する。これを図 8 に示す。図 8 に示す ρ 関数ブロックでは、1 ステージの乱数生成に必要なデータ生成のために 4 ク

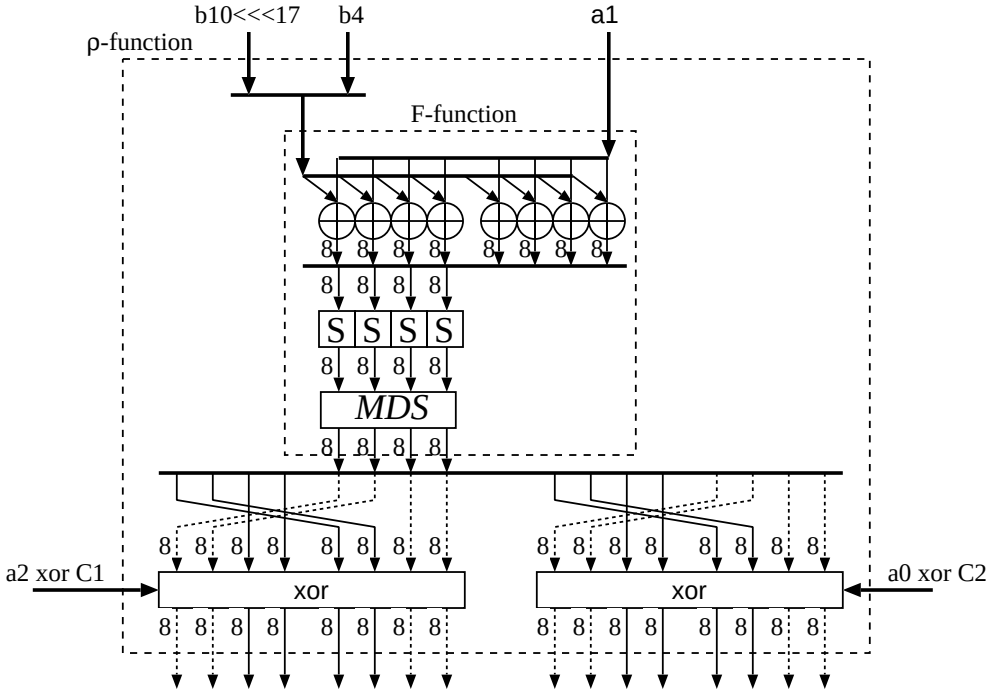


図 8: ρ 関数ブロック構成図 (小論理規模)

ロック必要である。つまり、速度優先の場合のおよそ 1/4 のスループットとなる。ρ 関数の入力 $a1$ の値は、ρ 関数の出力によって変化する。このため、あるラウンドの処理中に

入力 $a1$ の値が変化するのを防ぐために、$a1$ の値のコピーを保持するためのレジスタを用意する。S-box と MDS の部分は、32 ビットレジスタの追加でパイプライン化が可能である。パイプラインの段数は各段のデータ依存性により 3 段まで構成することができる。このパイプライン化により、見かけのゲート遅延が減少し、全体のスループットを向上できる。

上記のような方式で実装した結果を表 12 に示す。

表 12: ゲート規模 (小論理規模方式)

モジュール名	下位モジュールを含むゲート数	モジュール単体のゲート数
mugi(全体)	**18019**	**18019**
mugi	20702	14489
control	250	250
ρ	3774	572
F 関数	3202	215
MDS	307	307
S-box	670	670
init	181	181
λ	1826	1826

この実装方式で実装した場合、スループットは動作周波数 42.3MHz で 676Mbps となる。また、初期化には 4590ns を要する。ただし、この結果は 3 段パイプライン化を行っていない場合のものである。3 段パイプライン化の実装は現在のところ未評価であるが、実装した場合のゲート量の増加は 1K ゲート程度であり、スループットは動作周波数 126.6MHz で 2025Mbps、初期化の遅延は 1531ns 程度が見込まれる。

4.2.3 まとめ

以上のハードウェア実装に関する評価結果を 13 にまとめた。

表 13: ハードウェア実装のまとめ

実装方式	ゲート規模 (K gate)	動作周波数 (MHz)	スループット (Mbps)	初期化 (ns)
速度優先方式	26.1	45.7	2922	1095
小論理規模方式 (3 段パイプライン)	18.0 ($\geq$ 19.0)	42.3 (126.6)	676 (2025)	4590 (1531)

参考文献

[BS93] E. Biham, A. Shamir, "Differential Cryptanalysis of the Data Encryption Standard," Springer-Verlag, 1993.

[Da95] J. Daemen, "Cipher and hash function design strategies based on linear and differential cryptanalysis," Doctoral Dissertation, March 1995, K. U. Leuven.

[DC98] J. Daemen, C. Clapp, "Fast Hashing and Stream Encryption with PANAMA," *Fast Software Encryption*, Springer-Verlag, LNCS 1372, pp.60-74, 1998.

[DGV94] J. Daemen, R. Govaerts, J. Vandewalle, "Resynchronization weaknesses in synchronous stream ciphers," *Advances in Cryptology, Proceedings Eurocrypt'93*, Springer-Verlag, LNCS 765, pp.159-169, 1994.

[DKR97] J. Daemen, L. Knudsen, V. Rijmen, "The Block Cipher SQUARE," *Fast Software Encryption*, LNCS 1267, pp.149-165, Springer-Verlag, 1997.

[DR99] J. Daemen, V. Rijmen, "AES Proposal: Rijndael," AES algorithm submission, September 3, 1999, available at `http://www.nist.gov/aes/`.

[FIPS] "FIPS 140-1, Security Requirements for Cryptographic Modules," *Federal Information Processing Statndard (FIPS)*, Publication 140-1, National Institute of Standards adn Technology, US Department of Commerce, Washington D.C., January, 1994, available at `http://www.itl.nist.gov/fipspubs/index.htm`.

[JK97] T.Jacobsen and L.R. Knudsen, "The Interpolation Attack on Brock Ciphers," *Fast Software Encryption, FSE '97*, Springer-Verlag, LNCS 1267, pp.28-40, 1997.

[Kn81] D. Knuth, *The Art of Computer Programming Volume II (2nd ed.)*, Addison-Wesley, 1981.

[Ku94] L.R.Knudsen, "Truncated and Higher Order Differentials," *Fast Software En-*

cryption, FSE '94, Springer-Verlag, LNCS 1008, pp.196-211, 1995.

[Ma93] 松井　充, "DES 暗号の線形解読法 (I)," 暗号と情報セキュリティシンポジウム, SCIS 93-3C, 1993.

[MOV97] J. Menezes, C. van Oorschot, A. Vanstone, *HANDBOOK of APPLIED CRYPTOGRAPHY*, CRC Press, 1997.

[Ru86] R. A. Rueppel, *Analysis and Design of Stream Ciphers*, Springer-Verlag, 1986.

[Sc96] B. Schneier, *Applied Cryptography*, Second Edition, John Wiley & Sons, pp.397-398, 1996.

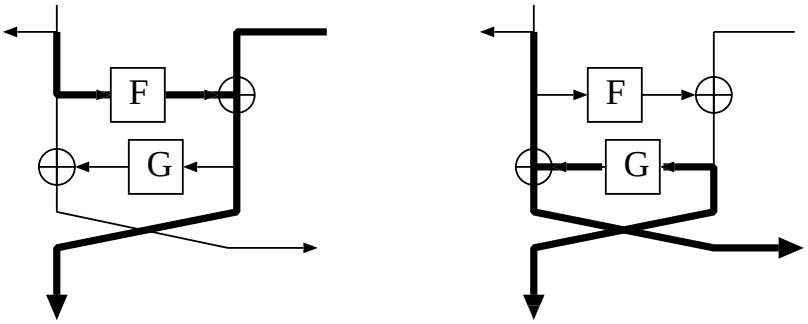
[WFT01] 渡辺 大, 古屋聡一, 宝木和夫, "F 関数を利用した鍵ストリーム生成器の設計法," 暗号と情報セキュリティシンポジウム, SCIS 2001-6A-4, 2001.

[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-, 2001.

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

A ρ 関数の線形パス



(F, G) = (1, 0) (F, G) = (0, 1)

図 9: ρ 関数の線形パス (1)

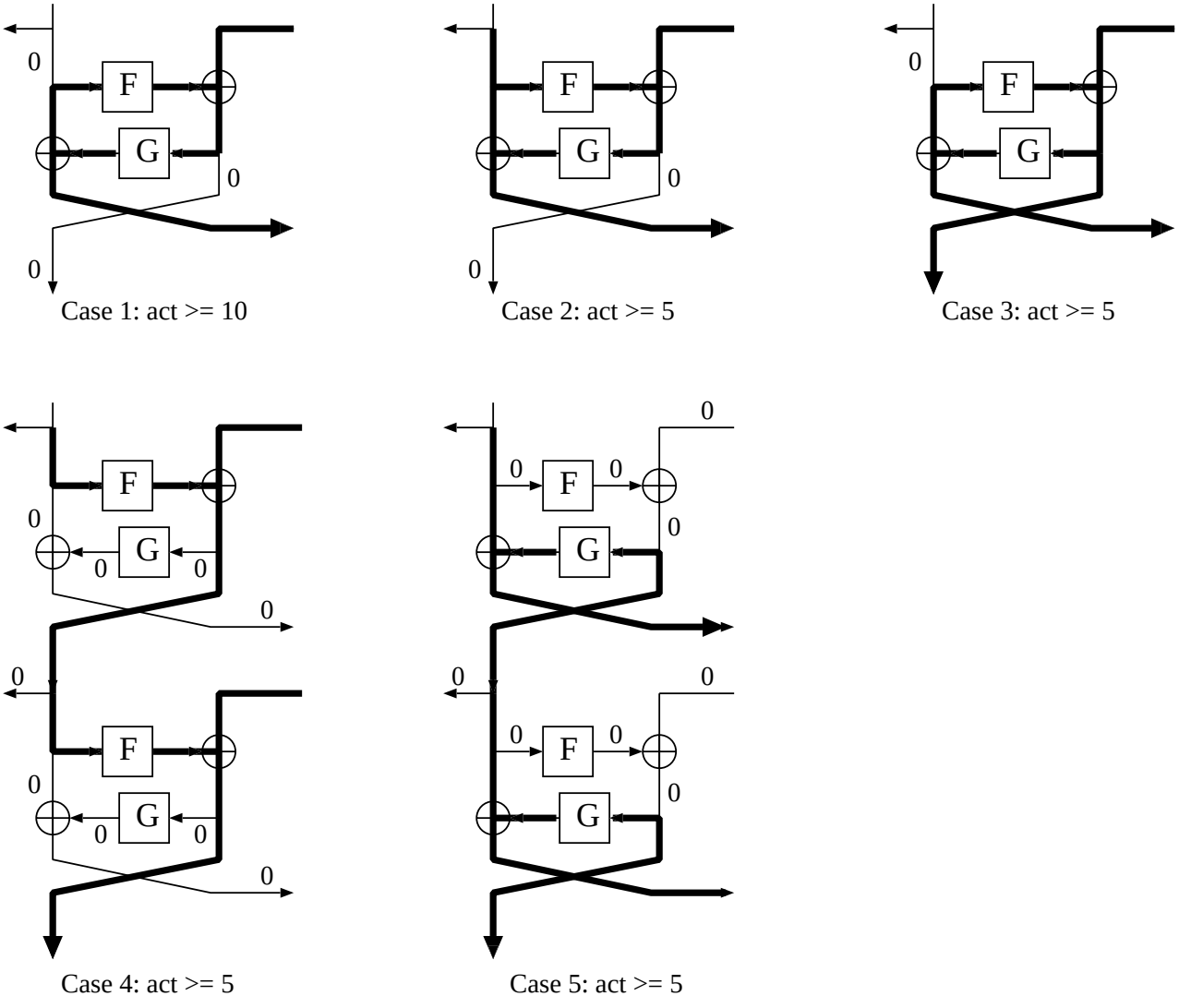


図 10: ρ 関数の線形パス (2)