

非線形処理構造を持つストリーム暗号に関する一検討

野間口 広^{1,a)} 岩切 宗利^{1,b)}

概要: 高速かつ大容量な暗号処理が可能な共通鍵ストリーム暗号は、将来のデジタル化社会を支える重要な情報セキュリティ技術の一つである。本研究では、標準化暗号の一つである K-Cipher2 と同等のリソースで実現可能であり、十分な暗号強度を持つ共通鍵ストリーム暗号の開発に取り組んでいる。本報告は、現在開発中の動的に変化する非線形処理構造を持つストリーム暗号の試作に関する設計方針とその基本処理について示し、それが鍵長よりも十分大きい複雑度を持ち、その出力乱数系列が統計的に優れた特性を持つことを示すものである。

キーワード: ストリーム暗号, LFSR, 非線形処理, S-Box, 動的 S-Box の更新

Abstract: Fast and high-capacity cryptographic processing is common key stream cipher possible, is one of the important information security technology to support the future of digital society. In this study, it can be realized in the K-Cipher2 equivalent of resources is one of the standard cipher, is working on the development of common key stream cipher with sufficient encryption strength. This report, shows design policies regarding prototype of stream cipher with a dynamically changing non-linear processing structure is currently under development and its basic processing, it has a long enough complexity than the key length, its output random number sequence it is indicated to have a statistically superior properties.

Keywords: Stream Cipher, LFSR, Non-Linear Processing, S-Box, Updating of Dynamic S-Box

1. はじめに

インターネットなどの高速かつ大容量な通信環境の整備と小型軽量な携帯型情報端末の普及により、マルチメディアを駆使した情報コンテンツをはじめ、あらゆる情報がいつでもどこでも利用できるユビキタス社会が実現しつつある。この時代背景のもと、情報の開示と非開示を適切に制御できる暗号の重要性が高まっており、その標準化も近年進んできている。

なかでも、暗号と復号に用いる鍵ストリーム(乱数系列)を事前に準備できるストリーム暗号は、その処理の軽量さと高速性から注目されており、将来のクラウドシステムやグローバルなコミュニケーションシステムで用いる情報セキュリティ技術として期待されている。

主なストリーム暗号の処理構造は、ブロック暗号利用型、状態遷移型、LFSR(LFSR:Linear Feedback Shift Register)利用型である。MUGI[1]は、ブロック暗号をベースに開

発されたもので、その内部状態を関数により遷移させながら、その一部を乱数列として取り出す方式を採用している。Py[2]は、秘密鍵から生成したテーブルを更新しながら、乱数列を生成する状態遷移型の方式を採用している。動的制御を施す複数の線形フィードバックレジスタとその出力系列の非線形変換部により、鍵となる乱数列を生成する方式は、eSTREAM[3]などでも数多く提案されている。

近年の動向としては、デジタル通信容量の増大やアプリケーションの多様化によって、より高い暗号処理速度が求められるようになり、2013年に改訂された電子政府[4]推奨暗号リストでは、高い計算量的安全性を保ちつつ、暗号処理の速い K-Cipher2 [5]のみが主要な推奨暗号として選出されている。LFSR 利用型の一方式である K-Cipher2 は、高速に動作する比較的少ないレジスタ数の LFSR を二つ持ち、そのフィードバック構造を動的に変化させる方式を採用している。その非線形変換部は、Substitution 関数(S 関数, S-Box)と Permutation 関数(P 関数)、ビット間干渉を用いた処理からなる。K-Cipher2 は、これらのシンプルかつ効率的な処理構造により、従来のストリーム暗号に比べてより少ないリソースで高速に処理できる。

本研究では、将来の情報化社会における情報セキュリティ

¹ 防衛大学校情報工学科
Department of Information Engineering, National Defense Academy,
Yokosuka, Kanagawa 239-8686, Japan

a) f14008@nda.ac.jp

b) iwak@nda.ac.jp

ティの確実性を高めるために、K-Cipher2と同等の性能を持つストリーム暗号の開発に取り組んでいる。本報告に示す試作暗号は、その最初の試みとして、K-Cipher2と同等のリソースで、異なる処理構造を持つLFSRと非線形処理を組み合わせて設計したものである。本試作暗号の乱数周期は、複数のLFSRを効率よく組み合わせることにより保証した。暗号強度の要である理論的複雑度は、非線形処理部に動的に変化する複数の非対称写像のS-Boxを用いることにより確保した。

本報告は、第2章に提案方式、第3章に処理ステップ、第4章に評価を示し、第5章に全体を総括する。

2. 提案方式

本研究は、計算量的安全性及び周期を保証しつつ、ストリーム暗号として重要な要件である高速さを満たすために図1の構造をとる。本研究で試作したストリーム暗号(以下、試作暗号)は、乱数源として長周期なLFSR群を持ち、乱数源より得た乱数系列の複雑度を向上する非線形変換部から構成される。本報告では、次のとおり記号表記する。

\oplus	:排他的論理和
$+$ (mod n)	: n 上における算術加算
\cdot	:論理積
$L1(n)$: n の下位 1 ビット
$L2(n)$: n の下位 2 ビット
$L4(n)$: n の下位 4 ビット
$L8(n)$: n の下位 8 ビット
$!$:階乗
\times	:積
\gg_n	:右向き n ビットシフト
\ll_n	:左向き n ビットシフト
<i>round</i>	:暗号鍵 32bit を出力するまでの工程
<i>clock()</i>	:() 内の LFSR の内部状態を 1 時刻進める
$LFSR(n)$:1 ビット n 段の Modular 型 LFSR
S-Box	:多表式非線形 S-Box
$S-Box[i]([j][k])$:S-Box の i 番目の表 (j 行, k 列)
S-Box-ch	:S-Box 更新関数
$chr(i, a, b)$:S-Box[i] の a 行と b 行の交換
$chc(i, a, b)$:S-Box[i] の a 列と b 列の交換
m_i	:32 ビットのメモリ (i はメモリの番号を示す)
M_i	:64 ビットのメモリ (i はメモリの番号を示す)
P	:Permutation 処理
t	:時刻 (t は任意の正数)
Z	:鍵ストリーム

2.1 アルゴリズムにおけるデータ構造

本研究では、暗号処理に用いるデータ構造を全て 32 ビット演算に最適化した。各レジスタの値は、基本的に 32 ビット

トを処理単位とした。本報告に示す値(ビット列やバイト値)は、左側を上位とする。

乱数源は、処理を高速化するため、多段の Modular 型 LFSR[6] で構成した。乱数源が持つ LFSR は、すべて M (Maximum) 系列を発生するように設計し、各段数を 95, 64, 63, 61 段とすることにより、生成される乱数列の長周期性を保証した。これらから得られるビット系列は、64 ビットの記憶装置 (M) に逐次記憶され、64 *clock* ごとに出力することにより、Modular 型 LFSR を効率よく利用できる。

非線形変換部は、出力系列から乱数源の内部状態を推測できないようにする役割を果たす。非線形変換部は、Permutation 処理 (P 関数)、記憶装置 (m)、多表式非線形 S-Box (S-Box) からなる。全体の処理構造は、1 時刻前の値をフィードバックしながら、動的係数処理により全体の複雑度が高くなるよう設計した。また、S-Box は 5 つの表からなり、各表とも入力に対する出力値の 0, 1 の出現頻度が均等になるように設計し、統計的に優れた乱数系列を出力するようにした。

制御部は、S-Box 更新関数及び m からなる。S-Box 更新関数は、各 S-Box の行、列のビット毎のバランスを崩さないように行と行、または列と列を交換する機能を持つ。1 時刻前の m_3 により P 関数は、制御する。

以上に述べた、乱数源、非線形変換部、制御部からなるストリーム暗号の構造を図 2 に示す。

P 関数は、 m_3 から 8 ビット取り出し、S-Box[4]を用い、乱数源の出力を図 3 のように攪拌処理を行う。この処理部では、乱数源 1 の出力ビット系列 256 ビットを 16 ビットのブロック単位で攪拌し、そのうち 192 ビットを非線形変換部の S-Box に渡し、残り 64 ビットを *round* 終了時の S-Box の更新及び次の時刻の P 関数処理に利用する。

非線形変換部内の m_1, m_2 は、32 ビットの値の一時記憶

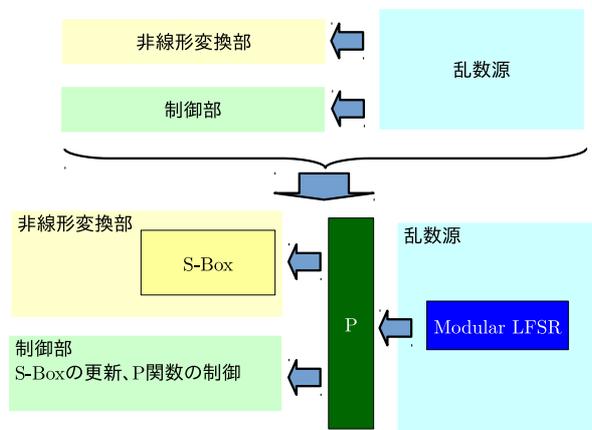


図 1 アルゴリズム概要
Fig. 1 Algorithm Overview

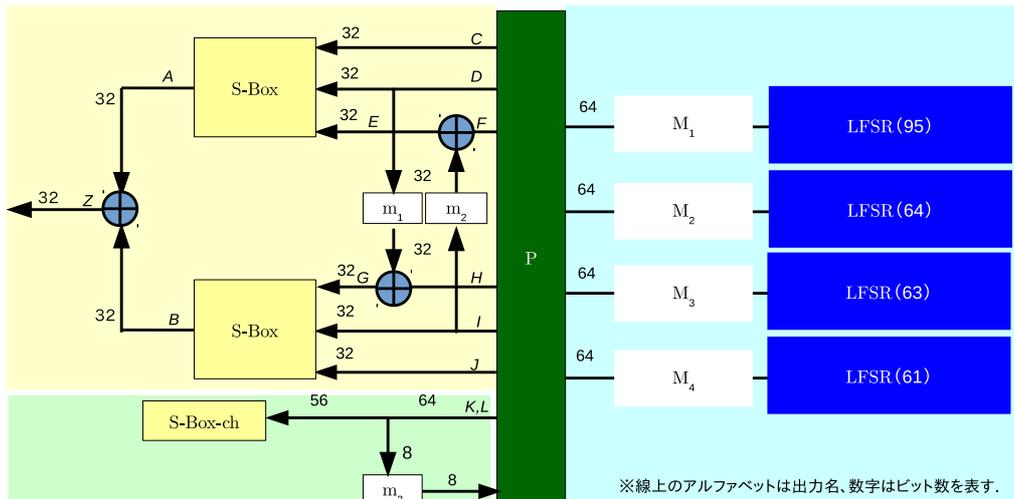
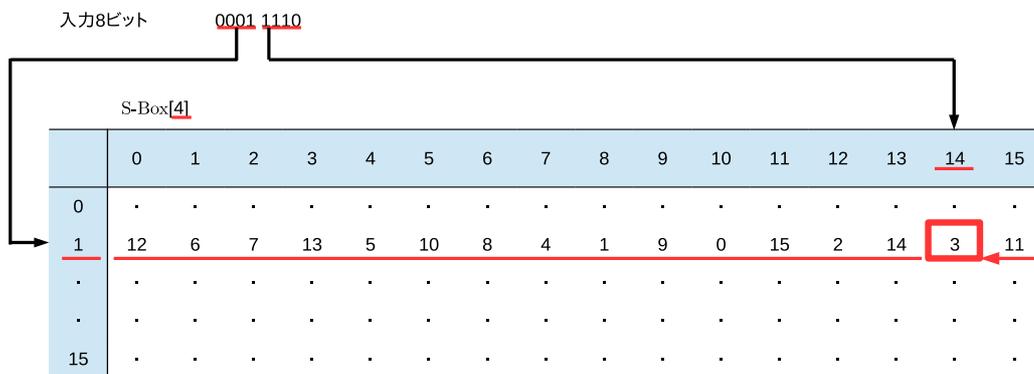


図 2 アルゴリズム
 Fig. 2 Algorithm



出力 3, 14, 2, 15, 0, 9, 1, 4, 8, 10, 5, 13, 7, 6, 12, 11

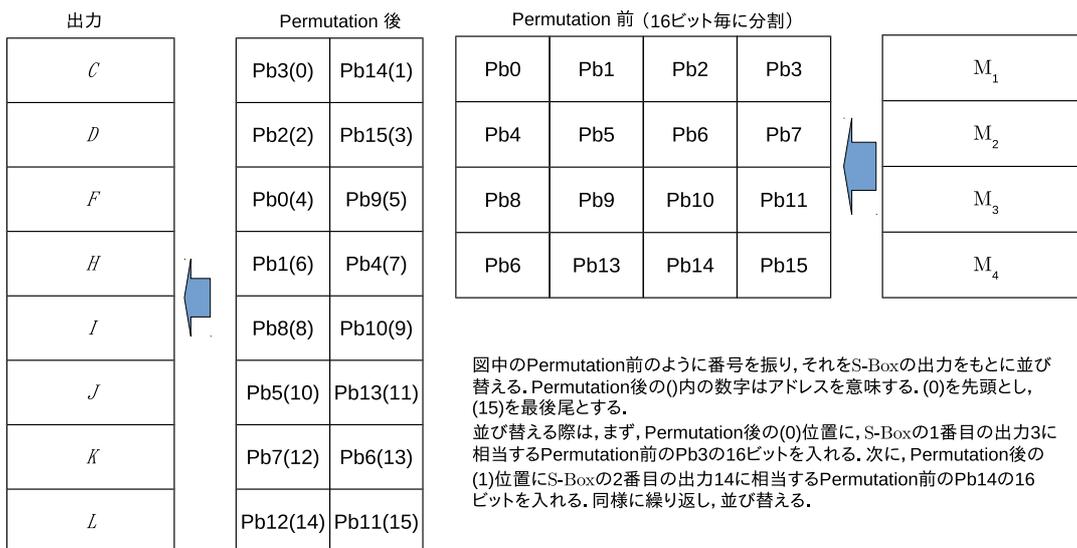


図 3 Permutation
 Fig. 3 Permutation

に用い、次の時刻での処理に利用することにより、出力乱数系列の理論的な複雑度を向上する。

S-Box は、32ビットの系列3個(合計96ビット)の入力値を32ビットに縮約する非線形かつ非対称な16行、16列、各要素4ビットの数値変換表5枚構成である。S-Boxの初期値は、図4のとおりである。S-Box[0]~S-Box[3]は非線形変換部の処理に、S-Box[4]はP関数に使用する。S-Box[0]~S-Box[3]は、入力として与えられた96ビットを12ビットの系列8個に分割し、各12ビットごとに4ビットに変換することにより、4ビット系列8個の値を得るものである。図5は12ビットの値を4ビット変換する処理である。S-Boxに与えた12ビットは、上位2ビットを表の選択に、4ビットを行指定に、下位6ビットを列指定に用い、4ビットを出力する。下位6ビットのビット列指定は、1行の各4ビットの数値16個を64ビットのビット列に見立てて、6ビットが示す位置から4ビットを抜きだし出力する。61ビット目以降は、次の行(最終行の場合は最初の行)の先頭ビットからとりだし、あわせて4ビット出力する。例えば、64ビット目を指定した場合、次の行の上位3ビットを加え、4ビットを出力する。

この非線形変換部の要であるS-Boxは、ビット値の出現頻度が均等になるように設計した。図4のS-Box[0]を例に説明する。1行目の各要素に着目すると、1~4ビット目に均等に0, 1が出現していることが分かる。これは、すべてのS-Boxの行、列に関して、同様に成り立つ。このため、入力の特定のビット列に、0または1が続いても、他方が均等に0, 1を入力しているならば、0, 1が均等に出力される。

また、S-Boxは、図6のように外部変数10ビットまたは8ビットにより行または列単位で動的に変化するように設計した。これは、ビット列の動的制御により出力ビット値の等頻度性を崩さない工夫である。S-Boxの更新のため図3で示したK, L(64ビット)のうち、上位40ビットはS-Box[0]~S-Box[3]の更新に、次の16ビットはS-Box[4]の更新に用いる。S-Box[0]~S-Box[3]に用いる40ビットは10ビット毎に分けられ、各10ビットのうち上位2ビットは、表番号の指定、下位8ビットは行(または列)の交換位置指定に用いる。16ビットのうち上位8ビットは行、下位8ビットは列の交換位置の指定に使用する。

3. 処理ステップ

本章では、32ビットの鍵ストリームの生成工程すなわち、1roundの処理を説明する。各工程は、鍵系列出力処理、S-Boxの更新、初期化処理に区分できる。

S-Box[0]																
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	12	4	2	7	5	15	14	13	10	0	8	6	1	9	3	11
1	5	0	8	15	7	9	13	1	14	6	3	11	4	2	12	10
2	2	14	7	3	8	12	6	11	0	13	15	1	10	5	9	4
3	3	1	9	5	12	7	10	14	11	4	2	0	13	15	8	6
4	13	5	11	4	1	0	8	3	2	7	10	15	12	6	14	9
5	1	7	10	0	4	6	3	12	8	15	14	9	5	11	13	2
6	9	10	5	8	2	3	0	6	4	14	7	13	11	12	15	1
7	8	13	15	12	3	5	11	10	6	1	9	4	14	7	2	0
8	6	2	1	10	11	14	7	15	5	8	4	3	9	13	0	12
9	11	8	4	14	10	13	15	9	7	3	0	12	2	1	6	5
10	0	9	13	11	6	10	5	7	12	2	1	8	15	14	4	3
11	10	3	0	13	14	1	9	2	15	12	6	5	8	4	11	7
12	7	6	3	9	15	2	1	4	13	11	12	10	0	8	5	14
13	15	11	12	2	9	8	4	0	1	10	5	14	6	3	7	13
14	14	12	6	1	13	4	2	8	9	5	11	7	3	0	10	15
15	4	15	14	6	0	11	12	5	3	9	13	2	7	10	1	8

S-Box[1]																
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	6	7	12	5	14	9	4	10	13	11	2	15	3	8	0	1
1	4	5	8	6	10	14	9	12	7	3	13	0	2	1	11	15
2	9	6	1	4	12	10	14	8	5	2	7	11	13	15	3	0
3	2	11	4	3	5	7	13	6	0	8	15	10	1	9	12	14
4	14	4	15	9	8	12	10	1	6	13	5	3	7	0	2	11
5	13	3	9	2	6	5	7	4	11	1	0	12	15	14	8	10
6	1	12	2	8	11	0	15	3	10	4	14	5	9	13	6	7
7	5	13	10	7	9	4	6	14	2	0	3	1	11	12	15	8
8	11	15	5	0	13	2	3	7	1	10	8	9	12	6	14	4
9	3	0	6	11	7	13	2	5	15	12	1	14	8	4	10	9
10	15	8	13	1	3	11	0	2	12	9	10	6	14	7	4	5
11	12	14	11	10	15	1	8	0	9	5	4	13	6	3	7	2
12	7	2	14	13	4	6	5	9	3	15	11	8	0	10	1	12
13	10	9	0	14	1	8	12	15	4	7	6	2	5	11	13	3
14	8	10	3	12	0	15	1	11	14	6	9	7	4	2	5	13
15	0	1	7	15	2	3	11	13	8	14	12	4	10	5	9	6

S-Box[2]																
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	4	0	15	9	13	8	6	11	10	5	7	12	14	3	1	2
1	15	7	12	5	11	13	1	2	8	4	3	0	6	9	10	14
2	13	14	11	10	5	9	0	4	3	8	6	2	12	1	7	15
3	6	8	1	2	0	12	5	7	15	14	13	10	9	11	4	3
4	0	9	7	15	14	2	8	6	11	12	5	3	10	4	13	1
5	7	5	3	12	6	14	13	1	2	0	4	9	8	15	11	10
6	2	1	14	13	15	4	3	12	5	11	10	6	7	8	9	0
7	14	10	6	11	12	15	9	0	4	2	8	1	3	13	5	7
8	5	12	4	3	8	10	14	13	1	9	0	15	2	7	6	11
9	12	3	0	4	2	11	10	14	13	15	9	7	1	5	8	6
10	8	2	13	1	9	3	12	5	7	10	14	11	15	6	0	4
11	1	13	10	14	7	0	4	3	12	6	11	8	5	2	15	9
12	3	4	9	0	1	6	11	10	14	7	15	5	13	12	2	8
13	10	11	8	6	3	7	15	9	0	1	2	13	4	14	12	5
14	9	15	5	7	10	1	2	8	6	3	12	4	11	0	14	13
15	11	6	2	8	4	5	7	15	9	13	1	14	0	10	3	12

S-Box[3]																
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	3	8	15	4	5	9	13	11	14	10	0	12	1	2	6	7
1	9	13	11	15	3	10	5	8	12	0	1	4	7	14	2	6
2	15	14	6	7	4	11	12	2	0	8	13	1	5	10	9	3
3	10	5	8	11	9	0	3	13	4	1	7	15	6	12	14	2
4	12	6	1	0	14	4	2	7	9	15	11	10	8	3	5	13
5	2	1	10	9	6	14	7	0	5	12	4	3	15	13	8	11
6	14	7	0	10	2	12	6	1	3	4	15	9	11	5	13	8
7	6	0	9	3	7	2	1	10	13	14	12	5	4	8	11	15
8	0	3	13	8	10	1	9	5	15	7	6	11	2	4	12	14
9	1	9	5	13	0	7	10	3	11	6	2	8	14	15	4	12
10	5	11	4	12	13	3	8	15	2	9	10	14	0	6	7	1
11	8	4	14	2	11	13	15	12	7	5	3	6	9	1	0	10
12	4	2	7	1	12	15	14	6	10	11	8	0	13	9	3	5
13	7	10	3	5	1	6	0	9	8	2	14	13	12	11	15	4
14	13	15	12	14	8	5	11	4	6	3	9	2	10	7	1	0
15	11	12	2	6	15	8	4	14	1	13	5	7	3	0	10	9

S-Box[4]																
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	15	12	4	3	14	8	6	11	1	9	13	10	7	0	2	5
1	12	2	14	9	6	15	1	4	10	5	11	0	3	13	7	8
2	2	7	6	5	1	12	10	14	0	8	4	13	9	11	3	15
3	0	13	5	14	8	10	15	9	12	6	3	2	4	7	11	1
4	7	3	1	8	10	2	0	6	13	15	14	11	5	4	9	12
5	6	1	7	13	3	14	9	2	5	11	12	8	0	15	10	4
6	11	4	15	1	12	13	2	8	7	10	5	3	6	9	14	0
7	4	14	12	10	2	11	7	15	3	0	8	9	1	5	6	13
8	3	9	10	15	0	7	13	1	11	12	6	4	8	14	5	2
9	5	8	13	2	11	9	4	0	14	7	10	6	12	1	15	3
10	14	6	2	0	7	4	3	12	9	13	15	5	10	8	1	11
11	13	11	8	6	15	0	12	5	2	1	9	7	14	3	4	10
12	1	10	3	11	9	6	5	7	8	4	2	15	13	12	0	14
13	9	5	0	12	13	3	11	10	4	2	1	14	15	6	8	7
14	10	0	9	4	5	1	8	3	15	14	7	12	11	2	13	6
15	8	15	11	7	4	5	14	13	6	3	0	1	2	10	12	9

図4 初期の S-Box
Fig. 4 Initial S-Box

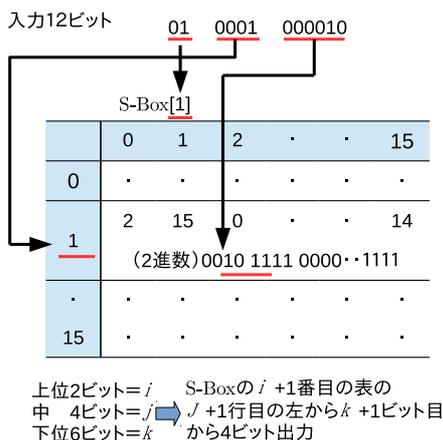
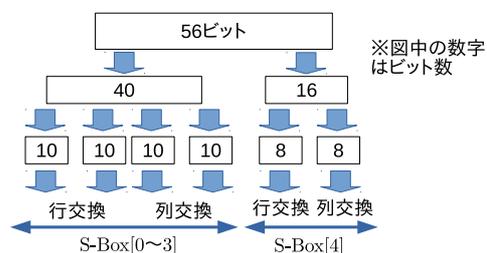
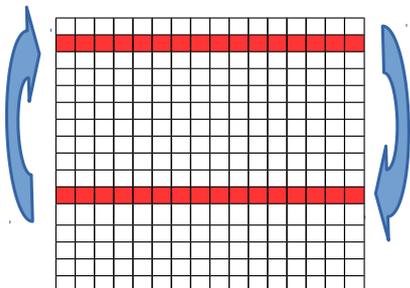


図 5 S-Box
Fig. 5 S-Box



chr関数 入力ビット 0001 1010 のとき
S-Boxの1行目と10行目の入れ替え



chc関数 入力ビット 0011 1101 のとき
S-Boxの3列目と13列目の入れ替え

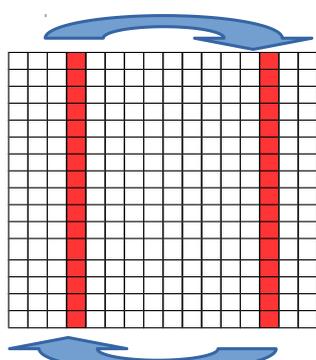


図 6 S-Box 更新要領
Fig. 6 S-Box update procedure

3.1 鍵系列出力処理

鍵系列出力処理は、乱数源、非線形変換からなる。

(1) 乱数源

```
for i=0 to 63 do {
  clock(LFSR(95))
  clock(LFSR(64))
  clock(LFSR(63))
  clock(LFSR(61))
  M1=M1 << 1+L1(LFSR(95))
  M2=M2 << 1+L1(LFSR(64))
  M3=M3 << 1+L1(LFSR(63))
  M4=M4 << 1+L1(LFSR(61))
}
```

(2) 非線形変換

非線形変換部は、乱数源の値を P 関数で置換したのちに、S-Box で変換する。P(i, j) は i (256 ビット), j (8 ビット) を用いた P 関数処理である。S-Box(C, D, F), S-Box(G, I, J) は、左の 32 ビットから、順に 12 ビットずつ取り出して 4 ビットに変換する処理である。

$C, D, F, H, I, J, K, L = P((M_1, M_2, M_3, M_4), m_3)$

$E = F \oplus m_2$

$G = H \oplus m_1$

$m_1 = D$

$m_2 = I$

$m_3 = L8(L)$

$A = S\text{-Box}(C, D, E)$

$B = S\text{-Box}(G, I, J)$

$Z = A \oplus B$

Z は 32 ビットの出力鍵ストリームである。

3.2 S-Box の更新要領

S-Box は、round 処理終了時に P 関数処理により得た K, L (各 32 ビット) を用いて更新する。

図 6 の更新関数は、chr(i, j, k) と chc(i, j, k) である。 i は表番号、 j と k は交換する行または列である。

chc(L2($K \gg 30$), L4($K \gg 26$), L4($K \gg 22$))

chc(L2($K \gg 20$), L4($K \gg 16$), L4($K \gg 12$))

chr(L2($K \gg 10$), L4($K \gg 6$), L4($K \gg 2$))

chr(L2(K), L4($L \gg 28$), L4($L \gg 24$))

chc(4, L4($L \gg 20$), L4($L \gg 16$))

chr(4, L4($L \gg 12$), L4($L \gg 8$))

3.3 初期化処理

鍵拡大、乱数源、S-Box の初期化に使用する初期鍵は、128 ビットである。初期化のアルゴリズムは、SHA-1 の手法を応用した。本方式では、SHA-1 の算術加算及び F 関数の代わりに S-Box を利用する。

3.3.1 鍵拡大

鍵拡大処理は、初期鍵と非線形変換部のアルゴリズムにより初期化に必要なビット列を得る。まず、初期鍵 K の

128ビットを K_1, K_2, K_3, K_4 の4つ(各32ビット)に分ける. さらに64ビットの $\{M_0, M_1, M_2\}$ を分割し, それぞれ32ビットの $\{m_{00}, m_{01}, m_{10}, m_{11}, m_{20}, m_{21}\}$ とする.

$$m_{00}=K_1, m_{01}=K_3+K_4(\text{mod } 2^{32})$$

$$m_{10}=K_2, m_{11}=K_3$$

$$m_{20}=K_1+K_2(\text{mod } 2^{32}), m_{21}=K_4$$

$$m_1=K_1+\overline{K_2} + \overline{K_4}(\text{mod } 2^{32})$$

$$m_2=K_4+\overline{K_1} + \overline{K_3}(\text{mod } 2^{32})$$

for $i=0$ to 9 do {

$$A=S\text{-Box}(m_{00}, m_{01}, m_{10}\oplus m_2)$$

$$B=S\text{-Box}(m_{11}\oplus m_1, m_{20}, m_{21})$$

$$Z_i=A\oplus B$$

$$m_1=m_{01}, m_2=m_{20}$$

$$m_{21}=m_{20}, m_{20}=m_{11}$$

$$m_{11}=m_{10}, m_{10}=m_{01}$$

$$m_{01}=m_{00}, m_{00}=Z_i$$

}

$\{Z_0, Z_1, Z_2\}$ の Z の値を LFSR(95), $\{Z_3, Z_4\}$ は LFSR(64), $\{Z_5, Z_6\}$ は LFSR(63), $\{Z_7, Z_8\}$ は LFSR(61), Z_9 は m_3 に順次格納する.

3.3.2 乱数源及び S-Box の初期化

初期化処理は, P 関数の処理を施さず, M を S-Box[0]~S-Box[3] の更新に, Z を S-Box[4] の更新に, それぞれ用いる.

for $k=0$ to 9 do {

for $i=0$ to 63 do {

$$\text{clock}(\text{LFSR}(95))$$

$$\text{clock}(\text{LFSR}(64))$$

$$\text{clock}(\text{LFSR}(63))$$

$$\text{clock}(\text{LFSR}(61))$$

$$M_1=M_1 \ll 1+L1(\text{LFSR}(95))$$

$$M_2=M_2 \ll 1+L1(\text{LFSR}(64))$$

$$M_3=M_3 \ll 1+L1(\text{LFSR}(63))$$

$$M_4=M_4 \ll 1+L1(\text{LFSR}(61))$$

}

$C, D, F, H, I, J, K, L=P((M_1, M_2, M_3, M_4), m_3)$

$$E=F\oplus m_2$$

$$G=H\oplus m_1$$

$$m_1=D$$

$$m_2=I$$

$$m_3=L8(L)$$

$$A=S\text{-Box}(C, D, E)$$

$$B=S\text{-Box}(G, I, J)$$

$$Z=A\oplus B$$

for $i=0$ to 3 do{

$$\text{chc}(0, L4(M_1), L4(M_1>>4))$$

$$\text{chr}(0, L4(M_1>>8), L4(M_1>>12))$$

$$\text{chc}(1, L4(M_2), L4(M_2>>4))$$

$$\text{chr}(1, L4(M_2>>8), L4(M_2>>12))$$

$$\text{chc}(2, L4(M_3), L4(M_3>>4))$$

$$\text{chr}(2, L4(M_3>>8), L4(M_3>>12))$$

$$\text{chc}(3, L4(M_4), L4(M_4>>4))$$

$$\text{chr}(3, L4(M_4>>8), L4(M_4>>12))$$

$$M_1=M_1>>16$$

$$M_2=M_2>>16$$

$$M_3=M_3>>16$$

$$M_4=M_4>>16$$

}

for $i=0$ to 1 do{

$$\text{chc}(4, L4(Z), L4(Z>>4))$$

$$\text{chr}(4, L4(Z>>8), L4(Z>>12))$$

$$Z=Z>>16$$

}

}

}

4. 評価

本章では, 本試作暗号の周期, 理論的暗号強度, 乱数特性について評価した結果を示す.

4.1 周期及び理論的暗号強度

周期は, 各 LFSR の各周期から理論的に求まる. 設計上の乱数周期は 2^{283} である.

次に理論的暗号強度について, 本検討では理論的暗号強

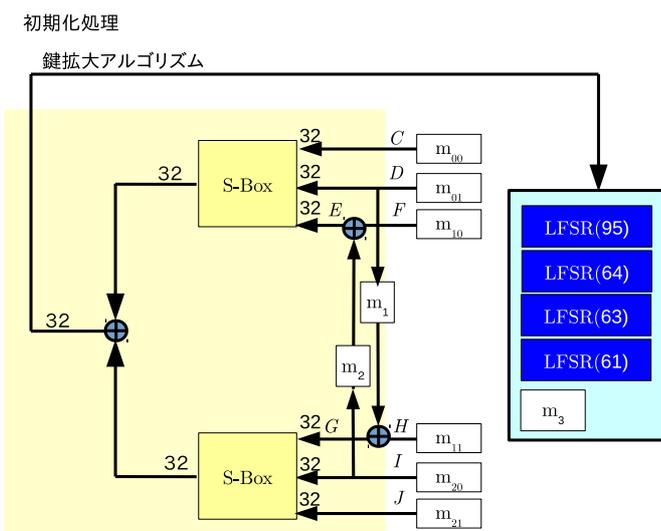


図 7 鍵拡大

Fig. 7 Key expansion

度を解読に要する計算量とする。試作暗号の出力系列を正しく予測するには乱数源と S-Box の内部状態を解読する必要がある。解読方法を、内部状態の総当たりする方法及び出力ビット列からの内部状態を求める方法の二つの観点から評価した。

(1) 総当たり計算量

内部状態の総当たりには要する計算量は、乱数源の LFSR, S-Box, 3つの m の総当たりが必要である。この乱数源の総当たりには要する計算量は、

$$2^{95} \times 2^{64} \times 2^{63} \times 2^{61} = 2^{283}$$

である。次に一つの S-Box に対する総当たりには要する計算量について考察する。一行に 4 ビットの要素が 16 個あり、この並び方は 16!通りある。2 行目は、列において 1 行目と同じ数字が入らないので 15!である。以降も同様に求めると一つの S-Box の内部状態を特定するには、

$$16! \times 15! \cdots \times 1 = 2^{300}$$

の計算が必要である。5つの S-Box の総当たりには要する計算量は

$$2^{300} \times 2^{300} \times 2^{300} \times 2^{300} \times 2^{300} = 2^{1500}$$

である。{m₁, m₂, m₃} の総当たりには要する計算量は、

$$2^{32} \times 2^{32} \times 2^8 = 2^{72}$$

である。よって、内部状態の総当たりには要する計算量は、

$$2^{283} \times 2^{300} \times 2^{1500} \times 2^{72} = 2^{2155}$$

である。

(2) 出力ビット列からの内部状態の推測に要する計算量

P 関数, S-Box の内部状態及びその処理, A と B の排他的論理和, 非線形変換部の m に分けて説明する。P 関数は、16 個の 16 ビットを並び替えているので計算量は

$$16 \times 15 \cdots \times 2 \times 1 = 2^{45}$$

である。S-Box は内部状態の総当たりには要する計算量に示したように 2¹⁵⁰⁰ の計算量が必要である。非線形変換部の S-Box 処理の計算量は 96 ビット入力に対し 32 ビット出力であるので、必要な計算量は 2⁶⁴ である。本提案方式は、非線形変換部に S-Box 処理を二つ用いているので

$$2^{64} \times 2^{64} = 2^{128}$$

である。時刻 t における S-Box の内部状態を把握した状態が分かっている状況から、時刻 t+1 における S-Box の内部状態を特定するのに必要な計算量について示す。S-Box[0]~S-Box[3] は行及び列の交換を 1round につき 2 回施される。表の選び方は 4 通り、行または列の交換位置の選び方は 16×16=2⁸ 通りである。これが各 round の更新ごとに 4 回行われるので、(2²×2⁸)⁴=2⁴⁰ 通りである。S-Box[4] の更新は、行、列の交換が 1 回ずつなので、必要計算量は (2⁸)²=2¹⁶ である。Z となる A と B の組み合わせ (A と B の排他的論理和の計算量) は 2³² 通りである。非線形変換部の m は総当たりによる内部状態を推定の計算量は 2³²×2³²=2⁶⁴ である。1 時刻前の出力から特定する場合は、A と B の排他的論理和, 非線形変換部の S-Box 処

STATISTICAL TEST	1回		2回		3回	
	VALUE	Pro	VALUE	Pro	VALUE	Pro
Frequency	0.010988	494/500	0.155499	499/500	0.002374	491/500
BlockFrequency CumulativeSums	0.013194	496/500	0.995162	493/500	0.171867	492/500
CumulativeSums	0.041169	494/500	0.220159	497/500	0.101311	489/500
Runs	0.169981	497/500	0.175691	494/500	0.325206	492/500
LongestRun	0.360287	492/500	0.666245	497/500	0.220159	496/500
Rank	0.786830	493/500	0.081510	494/500	0.003248	494/500
FFT	0.083526	495/500	0.408275	496/500	0.153763	490/500
NonOverlappingTemplate	0.002717	487/500	0.005204	489/500	0.005204	488/500
OverlappingTemplate	0.008506	488/500	0.000026	489/500	0.000830	487/500
Universal	0.812905	497/500	0.363593	496/500	0.628790	496/500
ApproximateEntropy	0.554420	494/500	0.131122	494/500	0.380407	497/500
RandomExcursions	0.267637	420/427	0.190726	423/430	0.004527	430/436
RandomExcursionsVariant	0.013411	421/427	0.181326	422/430	0.017347	426/436
Serial	0.743915	492/500	0.307077	496/500	0.183547	496/500
LinearComplexity	0.649612	495/500	0.173770	498/500	0.390721	499/500

※複数項目あるTESTはもっとも低かったものを記載
※VALUE:P-VALUE, Pro:PROPORTION

図 8 乱数検定結果

Fig. 8 Random number test result

理を行う必要があるので、必要計算量は

$$2^{32} \times 2^{64} / 3 \times 2^{64} / 3 = 2^{74}$$

である。このため、非線形変換部の内部状態を求めるには総当たりが最適である。

LFSR の内部状態の特定には段数の 2 倍以上の出力が必要であるため、出力から内部状態を確定するには複数 round に渡り上記の計算を行う必要がある。よって、出力ビット列からの内部状態を求めるには

$$2^{45} \times 2^{1500} \times 2^{128} \times 2^{40} \times 2^{16} \times 2^{32} \times 2^{64} = 2^{1825}$$

(1round の計算量) 以上の計算量が必要である。

4.2 乱数特性

NIST 評価ツールによる SP-800 22[7] を使用し、乱数検定を実施した。検定は、C 言語の SRAND 関数を使用し 128 ビットの鍵を生成したのちに 1 メガバイトの乱数を生成し、それを 500 本毎まとめて乱数検定を行った。図 8 は、乱数検定結果 3 回分である。図 8 中の赤字下線は、不合格を示す。

検定 1 回目は NonOverlappingTemplate の PROPORTION で 1 回、検定 2 回目は OverlappingTemplate の PROPORTION, 検定 3 回目は OverlappingTemplate の PROPORTION それぞれ不合格であった。

4.3 まとめ

周期は、鍵長より長く、理論的暗号強度は内部状態の総当たり、出力ビット列からの内部状態を求めるの二つの方法から計算量を算出したが、どちらも初期鍵 128 ビットの総当たり以上の計算量が必要であった。

統計的乱数検定では、1 メガバイト 500 本の検定を 3 回行ったところ 1 回の検定で 1 項目ずつ不合格となった。不合格となった TEST, 項目は異なっており、3 回の検定を通してみると特に問題ないと考えられる。このため本試作

暗号の統計的乱数特性は優れていると考える。

以上から、試作暗号の理論的暗号強度は鍵長より大きく、乱数周期及び統計的特性は優れていることが分かった。

5. 終わりに

本研究では、将来の情報化社会における情報セキュリティの確実性を高めるために、K-Cipher2と同等の性能を持つストリーム暗号の開発に取り組んだ。本報告に示した試作暗号は、電子政府推奨暗号に選ばれているK-Cipher2と同等のリソースで、異なる処理構造を持つLFSRと非線形処理を組み合わせて設計したものである。

本試作暗号は、複数のLFSRを効率よく組み合わせることにより乱数周期を保証し、暗号強度の要である理論的複雑度は、動的に変化する非対称写像のS-Boxを用いた非線形処理により確保した。

今後は、処理速度に着目して、K-Cipher2の性能を目標に開発する予定である。

参考文献

- [1] 株式会社 日立製作所: 疑似乱数生成器 MUGI 仕様書 Ver. 1.3 (2002年5月8日),
入手先 (http://www.cryptrec.go.jp/cryptrec_03_spec.cypherlist_files/PDF/10_02jspec.pdf) (2014.6.20).
- [2] Eli Biham., Jennifer Seberry.: *Py (Roo,):A Fast and Secure Stream Cipher using Rolling Arrays*, IACR Eprint archive2005/155(2005).
- [3] eSTREAM,the ECRYPT Stream Cipher Project,
<http://www.ecrypt.eu.org/stream/index.html>(2014.9.5).
- [4] 総務省, 経済産業省: 電子政府における調達のために参照すべき暗号のリスト (CRYPTREC暗号リスト),
入手先 (http://www.cryptrec.go.jp/images/cryptrec_ciphers_list_2013.pdf) (2014.6.20).
- [5] KDDI 株式会社:ストリーム暗号 KCipher-2,
入手先 (http://www.cryptrec.go.jp/cryptrec_13_spec.cypherlist_files/PDF/21_00jspec.pdf)(2014.6.20).
- [6] S.W. Golomb.:*Shift Register Sequence*, Aegean Park Press, Laguna Hills, California(1982).
- [7] NIST:*A Statistical Test Suite for Random and Pseudo-random Number Generators for Cryptographic Applications*, Special Publication 800-22 Revision 1a(2014.04) .