128ビットブロック暗号 CLEFIA

暗号技術仕様書

Version 1.0

ソニー株式会社

平成22年1月29日

変更履歴

Jan 29, 2010 version 1.0

目 次

第1章　設計方針

1.1　CLEFIA の設計方針

ブロック暗号の設計解析技術に発展にともない，AES [5] をはじめ，高い安全性と高い性能を備えた多くのブロック暗号が提案されてきた．しかしながら，攻撃法は日々進化しており，近年では代数攻撃 [4] や関連鍵攻撃などが進化を遂げ [1–3]，これらの攻撃法に配慮した設計が求められている．また，暗号技術の適用領域はますます拡大しており，スマートカードや RFID など，制約条件の厳しい環境下でも実装可能な暗号技術の需要が高まっている．

このようなニーズに応えるべく，我々は最新の研究成果や設計手法に基づきブロック暗号 CLEFIA を設計した．CLEFIA はブロック長 128 ビット，鍵長 128, 192, 256 ビットに対応しており，AES の入出力仕様とも互換性がある．

CLEFIA の設計方針は，実用的な暗号に求められる 3 つの基本要素

- 安全性
- 速度
- 実装コスト

をバランスよく実現することである．これらの目的を達成するため，CLEFIA にはいくつかの新しいアイデアが取り入れられている．これらのアイデアを以下に示す．

全体構造　CLEFIA では全体構造として，4 系列一般化 Feistel 構造を採用している．一般化 Feistel 構造にも様々な種類があるが，CLEFIA はその中でも特に Zheng らによって定義された Type-2 一般化 Feistel 構造を採用している [12]．4 系列の Type-2 一般化 Feistel 構造は，1 ラウンドに 2 つの F 関数を持ち，以下に挙げる特徴を持つ．

- F 関数のサイズは通常の Feistel 構造の半分である
- 複数の F 関数が同時に実行できる

- 通常の Feistel 構造と比較して多くのラウンド数が必要

1 つ目の特長は，ソフトウェア，およびハードウェア実装において大きなアドバンテージとなる．2 つ目の特長も効率的な実装に寄与し，特にハードウェア実装において実用性を大きく高めることに貢献している．以上より我々は，3 つ目の短所を含めても Type-2 一般化 Feistel 構造は通常の Feistel 構造と比較して有利な点が多いと考えた．

さらに CLEFIA では，次に説明する拡散行列切り替え法 (DSM) と呼ばれる設計技法を用いることで，必要なラウンド数を削減することに成功している．このため，全体構造として効率の高い構造となっている．

拡散行列切り替え法 (DSM)　CLEFIA の新規設計方針の一つは，拡散行列切り替え法 (DSM) [7,8] の採用である．拡散行列切り替え法とは，F 関数で用いられる拡散行列として 2 つ以上の異なる拡散行列から暗号化関数内の位置に応じて選択することで差分・線形攻撃への耐性を高める手法である．この手法により， 必要なラウンド数を削減することに成功している．

2 種類の S-box　CLEFIA では，異なる代数構造に基づく 2 種類の S-box を採用し，代数攻撃に対する耐性の向上をはかっている．

安全かつコンパクトな鍵スケジュール　CLEFIA では，鍵スケジュールでも新しい設計を取り入れている．鍵スケジュール部の処理にはデータ処理部と同じ一般化 Feistel 構造を使用することで，データ処理部との部品共有化を可能にした．さらにこのような構造を採用することで，安全性解析も容易となり，特に近年注目を集めている関連鍵攻撃に対しての評価がなされている．また，鍵スケジュール部で用いられている $DoubleSwap$ 関数は，拡散性能を保ちつつ軽量な構造となっており，暗号化時，復号時ともに，中間鍵を生成することで，単純な処理で逐次的にラウンド鍵を生成するようなハードウェア実装も容易にしている．

高い実装性能をめざした設計　ハードウェア，ソフトウェア問わず効率的な実装が可能となるよう，CLEFIA には表 1.1 に示すような特長が盛り込まれている．

1.2　CLEFIA の特長

本節では，公募要項に基づき，電子政府推奨暗号リスト (現リスト) に記載された暗号技術と同等以上の特長について記述する．

表 1.1: 実装効率に関する CLEFIA の特長

一般化 Feistel 構造	· 小さな F 関数 (32 ビット入出力) · F 関数の逆関数は不要
SP 型 F 関数	· 効率的なテーブル実装が可能 (ソフトウェア実装時)
DSM	· ラウンド数の削減が可能
S-box	· コンパクト実装に適した S_0, S_1 (特にハードウェア実装時)
行列	· 要素のハミングウェイトが小さい
鍵スケジュール部	· データ処理部と共有可能 · 128 ビット鍵の場合，必要なレジスタは 128 ビットレジスタ 1 つのみ · コンパクトな $DoubleSwap$ 関数

既存攻撃法に対する安全性確認　共通鍵ブロック暗号に対して，設計時に知られている攻撃法については網羅的に取り上げ，それぞれについて配慮した設計を行い，安全性評価において問題がないことを確認している．

特に，CLEFIA では，2 つの異なる拡散行列を用いて差分攻撃法および線形攻撃法への耐性を高める新しい設計技法「拡散行列切り替え法 (DSM)」を採用し，差分攻撃法および線形攻撃法に対する定量的な安全性評価を示している．

進化する攻撃法への対応　共通鍵ブロック暗号に対する攻撃法は日々進化しており，CLEFIA の設計にあたっては，現リスト記載のブロック暗号の設計時点からのさまざまな暗号解読法の進歩が考慮されている．

特に，近年，関連鍵攻撃の進展がめざましく，AES 等のシンプルな鍵スケジュールをもつブロック暗号への適用が進んでいる．CLEFIA では鍵スケジュール部に，データ処理部の構造と同じ Type-2 一般化 Feistel 構造を採用し，鍵スケジュール部単体でも高い安全性 (差分攻撃法および線形攻撃法に対する高い耐性) を保証しており，このような攻撃の適用が困難な設計となっている．

さらに，代数的に異なる構造をもつ 2 種類の S-box を F 関数内に配置することで， XSL 攻撃等の代数攻撃を含む各種攻撃に対する安全性を高めている．

高い実装性能　CLEFIA は最新の暗号解析技術に基づいた高い安全性を持ちつつ，ハードウェア，ソフトウェア問わず高い実装性能をも併せ持つ．特にハードウェア実装においては顕著な性能を発揮する．

CLEFIA のソフトウェア実装性能は，クロック周波数 2.4GHz の AMD Athlon 64 プロセッサで，13cycles/byte，1.48Gbps を達成している．これは現リストに記載されたブロック暗号技術の最も高速なグループに属すると考えられる．

ハードウェア実装性能は，0.09μm CMOS 標準セルライブラリを使用した場合に ハードウェア規模 5Kgate 以下で実装することが可能であり，これは現リストに記載されたブロック暗号技術の最も小型実装が可能なグループに属する．

また，より高速性を追求した実装では，約 6Kgate で 1.6Gps，約 12Kgate で 3Gbps を超える高速性を達成することが可能で，単位ゲート数あたりの処理速度で見ると，現リストに記載されたブロック暗号技術を超える性能を示している．

2007 年に菅原らによって ISO/IEC 18033-3 に採択されている標準ブロック暗号との ASIC ハードウェア性能比較が行われているが，CLEFIA の回路効率 (スループット/回路面積) について優位性があることが示されている [10, 11]．

第2章　アルゴリズム仕様

本章ではブロック暗号 CLEFIA の仕様について記述する．CLEFIA は鍵長 128，192，256 ビットに対応した 128 ビットブロック暗号であり，AES の入出力仕様と互換性がある．CLEFIA はデータ処理部と鍵スケジュール部の 2 つの処理部から構成される．CLEFIA は一般化 Feistel 構造を採用し，各系列のデータサイズは 32bit である．またデータ処理部の初期及び最終処理には鍵ホワイトニング部を持つ．CLEFIA のラウンド数は鍵長 128，192，256 ビットに対してそれぞれ 18，22，26 である．

2.1 表記方法について

本節では以下に述べる記号，表記を用いる．

`0x`	:	16 進数表記の prefix
$a_{(b)}$	:	b は a のビットサイズ
$a\|b$ or $(a\|b)$	:	連結
$(a,\ b)$ or $(a\ b)$	:	$a\|b$ のベクトル表現
$a \leftarrow b$	:	a に b を代入
${}^t a$	:	行列，またはベクトル a の転置
$a \oplus b$	:	ビットごとの排他的論理和 (GF(2^n) 上の加算)
$a \cdot b$	:	GF(2^n) 上の乗算
$\overline{a}$	:	a のビット反転
$a \lll b$	:	b ビット左循環シフト

2.2 $GFN_{d,r}$ の定義

本節では，まず CLEFIA の基本構成となる関数 $GFN_{d,r}$ を定義し，これを用い，データ処理部，及び鍵スケジュール部の説明を行う．

CLEFIA では 4 系列と 8 系列の一般化 Feistel 構造を使用する．CLEFIA で用いる d 系列，r ラウンドの一般化 Feistel 構造を $GFN_{d,r}$ と表記し，$GFN_{d,r}$ は以下で定義される 2 つの異なる 32 ビット入出力 F 関数 F_0 と F_1 を用いる．

$$F_0, F_1 : \begin{cases} \{0,1\}^{32} \times \{0,1\}^{32} & \rightarrow \quad \{0,1\}^{32} \\ (RK_{(32)}, x_{(32)}) & \mapsto \quad y_{(32)} \end{cases}$$

d 個の 32 ビット入力 X_i と出力 Y_i $(0 \leq i < d)$，及び $dr/2$ 個の 32 ビットのラウンド鍵 RK_i $(0 \leq i < dr/2)$ を用い，$GFN_{d,r}$ $(d = 4, 8)$ は以下のように定義される．

$$GFN_{4,r} : \begin{cases} \{\{0,1\}^{32}\}^{2r} \times \{\{0,1\}^{32}\}^4 \rightarrow \{\{0,1\}^{32}\}^4 \\ (RK_{0(32)}, \ldots, RK_{2r-1(32)}, X_{0(32)}, \ldots, X_{3(32)}) \mapsto Y_{0(32)}, \ldots, Y_{3(32)} \end{cases}$$

Step 1.	$T_0 \mid T_1 \mid T_2 \mid T_3 \leftarrow X_0 \mid X_1 \mid X_2 \mid X_3$	
Step 2.	$i = 0$ から $r-1$ に対して以下を実行：	
	Step 2.1	$T_1 \leftarrow T_1 \oplus F_0(RK_{2i}, T_0)$,
		$T_3 \leftarrow T_3 \oplus F_1(RK_{2i+1}, T_2)$
	Step 2.2	$T_0 \mid T_1 \mid T_2 \mid T_3 \leftarrow T_1 \mid T_2 \mid T_3 \mid T_0$
Step 3.	$Y_0 \mid Y_1 \mid Y_2 \mid Y_3 \leftarrow T_3 \mid T_0 \mid T_1 \mid T_2$	

$$GFN_{8,r} : \begin{cases} \{\{0,1\}^{32}\}^{4r} \times \{\{0,1\}^{32}\}^8 \rightarrow \{\{0,1\}^{32}\}^8 \\ (RK_{0(32)}, \ldots, RK_{4r-1(32)}, X_{0(32)}, \ldots, X_{7(32)}) \mapsto Y_{0(32)}, \ldots, Y_{7(32)} \end{cases}$$

Step 1.	$T_0 \mid T_1 \mid \ldots \mid T_7 \leftarrow X_0 \mid X_1 \mid \ldots \mid X_7$	
Step 2.	$i = 0$ から $r-1$ に対して以下を実行：	
	Step 2.1	$T_1 \leftarrow T_1 \oplus F_0(RK_{4i}, T_0)$,
		$T_3 \leftarrow T_3 \oplus F_1(RK_{4i+1}, T_2)$,
		$T_5 \leftarrow T_5 \oplus F_0(RK_{4i+2}, T_4)$,
		$T_7 \leftarrow T_7 \oplus F_1(RK_{4i+3}, T_6)$
	Step 2.2	$T_0 \mid T_1 \mid \ldots \mid T_6 \mid T_7 \leftarrow T_1 \mid T_2 \mid \ldots \mid T_7 \mid T_0$
Step 3.	$Y_0 \mid Y_1 \mid \ldots \mid Y_6 \mid Y_7 \leftarrow T_7 \mid T_0 \mid \ldots \mid T_5 \mid T_6$	

$GFN_{d,r}$ の逆関数である $GFN^{-1}_{4,r}$ は，ラウンド鍵 RK_i の順序，及び Step 2.2, Step 3 におけるワード巡回の方向を入れ替えることで定義することができる．

$$GFN^{-1}_{4,r}: \begin{cases} \{\{0,1\}^{32}\}^{2r} \times \{\{0,1\}^{32}\}^{4} \rightarrow \{\{0,1\}^{32}\}^{4} \\ (RK_{0(32)}, \ldots, RK_{2r-1(32)}, X_{0(32)}, \ldots, X_{3(32)}) \mapsto Y_{0(32)}, \ldots, Y_{3(32)} \end{cases}$$

Step 1.	$T_0 \mid T_1 \mid T_2 \mid T_3 \leftarrow X_0 \mid X_1 \mid X_2 \mid X_3$	
Step 2.	$i = 0$ から $r-1$ に対して以下を実行：	
	Step 2.1	$T_1 \leftarrow T_1 \oplus F_0(RK_{2(r-i)-2}, T_0)$,
		$T_3 \leftarrow T_3 \oplus F_1(RK_{2(r-i)-1}, T_2)$
	Step 2.2	$T_0 \mid T_1 \mid T_2 \mid T_3 \leftarrow T_3 \mid T_0 \mid T_1 \mid T_2$
Step 3.	$Y_0 \mid Y_1 \mid Y_2 \mid Y_3 \leftarrow T_1 \mid T_2 \mid T_3 \mid T_0$	

2.2.1 F 関数

$GFN_{d,r}$ で用いられる 2 つの F 関数 F_0 及び F_1 は以下のように定義される．

$F_0 : (RK_{(32)}, x_{(32)}) \mapsto y_{(32)}$

Step 1. $T \leftarrow RK \oplus x$

Step 2. $T = T_0 \mid T_1 \mid T_2 \mid T_3,\ T_i \in \{0,1\}^8$ とする

$T_0 \leftarrow S_0(T_0)$,

$T_1 \leftarrow S_1(T_1)$,

$T_2 \leftarrow S_0(T_2)$,

$T_3 \leftarrow S_1(T_3)$

Step 3. $y = y_0 \mid y_1 \mid y_2 \mid y_3,\ y_i \in \{0,1\}^8$ とする

${}^t(y_0, y_1, y_2, y_3) = M_0\ {}^t(T_0, T_1, T_2, T_3)$

$F_1 : (RK_{(32)}, x_{(32)}) \mapsto y_{(32)}$

Step 1. $T \leftarrow RK \oplus x$

Step 2. $T = T_0 \mid T_1 \mid T_2 \mid T_3,\ T_i \in \{0,1\}^8$ とする

$T_0 \leftarrow S_1(T_0)$,

$T_1 \leftarrow S_0(T_1)$,

$T_2 \leftarrow S_1(T_2)$,

$T_3 \leftarrow S_0(T_3)$

Step 3. $y = y_0 \mid y_1 \mid y_2 \mid y_3,\ y_i \in \{0,1\}^8$ とする

${}^t(y_0, y_1, y_2, y_3) = M_1\ {}^t(T_0, T_1, T_2, T_3)$

S_0，S_1 はそれぞれ 8 ビットの入出力の S-box を表し，M_0，M_1 はそれぞれ 4×4 の行列を表している．この S-box 及び行列の定義については後の節にて説明する．2 つの F 関数 F_0，F_1 において，2 つの S-box の配置順は異なっている．また行列についても異なるものが用いられている．図 2.1 に F 関数の構成を示す．

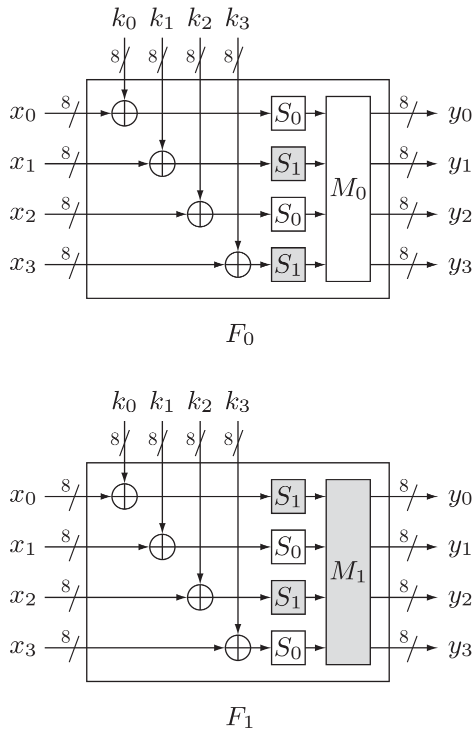

図 2.1: F 関数

2.2.2 S-box

CLEFIA は 2 つの異なる S-box を採用している．ひとつは，ランダムに選択された 4 つの 4 ビット入出力 S-box をベースとした S-box であり，もうひとつは，$GF(2^8)$ 上の逆元関数をベースとした S-box である．

表 2.1 及び 2.2 は，各 S-box S_0，S_1 の入出力を示している．これらテーブルの入出力値は 16 進数で表現されており，8 ビットの入力に対して，上位 4 ビットがテーブルの行，下位 4 ビットがテーブルの列に対応する．例えば，テーブル S_0 の場合，`0xab` が S-box への入力とすると，行の値が'`a.`'，列の値が'`.b`' となるため，出力は `0x7e` はとなる．

表 2.1: S_0

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.a	.b	.c	.d	.e	.f
0.	57	49	d1	c6	2f	33	74	fb	95	6d	82	ea	0e	b0	a8	1c
1.	28	d0	4b	92	5c	ee	85	b1	c4	0a	76	3d	63	f9	17	af
2.	bf	a1	19	65	f7	7a	32	20	06	ce	e4	83	9d	5b	4c	d8
3.	42	5d	2e	e8	d4	9b	0f	13	3c	89	67	c0	71	aa	b6	f5
4.	a4	be	fd	8c	12	00	97	da	78	e1	cf	6b	39	43	55	26
5.	30	98	cc	dd	eb	54	b3	8f	4e	16	fa	22	a5	77	09	61
6.	d6	2a	53	37	45	c1	6c	ae	ef	70	08	99	8b	1d	f2	b4
7.	e9	c7	9f	4a	31	25	fe	7c	d3	a2	bd	56	14	88	60	0b
8.	cd	e2	34	50	9e	dc	11	05	2b	b7	a9	48	ff	66	8a	73
9.	03	75	86	f1	6a	a7	40	c2	b9	2c	db	1f	58	94	3e	ed
a.	fc	1b	a0	04	b8	8d	e6	59	62	93	35	7e	ca	21	df	47
b.	15	f3	ba	7f	a6	69	c8	4d	87	3b	9c	01	e0	de	24	52
c.	7b	0c	68	1e	80	b2	5a	e7	ad	d5	23	f4	46	3f	91	c9
d.	6e	84	72	bb	0d	18	d9	96	f0	5f	41	ac	27	c5	e3	3a
e.	81	6f	07	a3	79	f6	2d	38	1a	44	5e	b5	d2	ec	cb	90
f.	9a	36	e5	29	c3	4f	ab	64	51	f8	10	d7	bc	02	7d	8e

表 2.2: S_1

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.a	.b	.c	.d	.e	.f
0.	6c	da	c3	e9	4e	9d	0a	3d	b8	36	b4	38	13	34	0c	d9
1.	bf	74	94	8f	b7	9c	e5	dc	9e	07	49	4f	98	2c	b0	93
2.	12	eb	cd	b3	92	e7	41	60	e3	21	27	3b	e6	19	d2	0e
3.	91	11	c7	3f	2a	8e	a1	bc	2b	c8	c5	0f	5b	f3	87	8b
4.	fb	f5	de	20	c6	a7	84	ce	d8	65	51	c9	a4	ef	43	53
5.	25	5d	9b	31	e8	3e	0d	d7	80	ff	69	8a	ba	0b	73	5c
6.	6e	54	15	62	f6	35	30	52	a3	16	d3	28	32	fa	aa	5e
7.	cf	ea	ed	78	33	58	09	7b	63	c0	c1	46	1e	df	a9	99
8.	55	04	c4	86	39	77	82	ec	40	18	90	97	59	dd	83	1f
9.	9a	37	06	24	64	7c	a5	56	48	08	85	d0	61	26	ca	6f
a.	7e	6a	b6	71	a0	70	05	d1	45	8c	23	1c	f0	ee	89	ad
b.	7a	4b	c2	2f	db	5a	4d	76	67	17	2d	f4	cb	b1	4a	a8
c.	b5	22	47	3a	d5	10	4c	72	cc	00	f9	e0	fd	e2	fe	ae
d.	f8	5f	ab	f1	1b	42	81	d6	be	44	29	a6	57	b9	af	f2
e.	d4	75	66	bb	68	9f	50	02	01	3c	7f	8d	1a	88	bd	ac
f.	f7	e4	79	96	a2	fc	6d	b2	6b	03	e1	2e	7d	14	95	1d

表 2.3: SS_i $(0 \leq i < 4)$

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$SS_0(x)$	e	6	c	a	8	7	2	f	b	1	4	0	5	9	d	3
$SS_1(x)$	6	4	0	d	2	b	a	3	9	c	e	f	8	7	5	1
$SS_2(x)$	b	8	5	e	a	6	4	c	f	7	2	3	1	0	d	9
$SS_3(x)$	a	2	6	d	3	4	5	e	0	7	8	9	b	f	c	1

S_0 の定義

S_0 は 4 つの 4 ビット入出力 S-box SS_0, SS_1, SS_2, SS_3 を用いて以下のように定義される．各 S-box の入出力仕様を表 2.3 に示す．

$$S_0 : \begin{cases} \{0,1\}^8 & \rightarrow & \{0,1\}^8 \\ x_{(8)} & \mapsto & y_{(8)} \end{cases}$$

Step 1. $t_0 \leftarrow SS_0(x_0),\ t_1 \leftarrow SS_1(x_1),$ 但し $x = x_0|x_1,\ x_i \in \{0,1\}^4$
Step 2. $u_0 \leftarrow t_0 \oplus \texttt{0x2} \cdot t_1,\ u_1 \leftarrow \texttt{0x2} \cdot t_0 \oplus t_1$
Step 3. $y_0 \leftarrow SS_2(u_0),\ y_1 \leftarrow SS_3(u_1),$ 但し $y = y_0|y_1,\ y_i \in \{0,1\}^4$

アルゴリズム中の乗算 $\texttt{0x2} \cdot t_i$ は，辞書的順序で最初となる原始多項式 $z^4 + z + 1$ で定義される GF(2^4) 上の演算として実行される．図 2.2 に S_0 の構成を図示する．

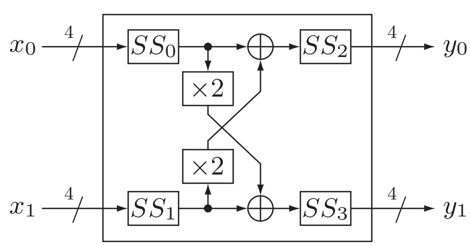

図 2.2: S_0

S_1 の定義

S_1 は以下のように定義される．

$$S_1 : \begin{cases} \{0,1\}^8 & \rightarrow & \{0,1\}^8 \\ x_{(8)} & \mapsto & y_{(8)} \end{cases}$$

$$\boxed{y = \begin{cases} g(f(x)^{-1}) & f(x) \neq 0 \text{ の場合} \\ g(0) & f(x) = 0 \text{ の場合} \end{cases}}$$

ここで逆元関数は原始多項式 $z^8 + z^4 + z^3 + z^2 + 1$ で定義される GF(2^8) 上の演算として定義される．また $f(\cdot)$ と $g(\cdot)$ は GF(2) 上のアフィン変換として以下のように定義される．

$$f : \begin{cases} \{0,1\}^8 & \rightarrow & \{0,1\}^8 \\ x_{(8)} & \mapsto & y_{(8)} \end{cases}$$

$$\boxed{\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} 0&0&0&1&1&0&0&0 \\ 0&1&0&1&0&0&0&1 \\ 0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&1&1&0 \\ 0&1&1&0&0&1&0&1 \\ 0&1&0&1&1&1&0&0 \\ 0&1&1&0&0&0&0&0 \\ 1&0&0&0&0&0&0&1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}}$$

$$g : \begin{cases} \{0,1\}^8 & \rightarrow & \{0,1\}^8 \\ x_{(8)} & \mapsto & y_{(8)} \end{cases}$$

$$\boxed{\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} 0&0&0&0&1&0&1&0 \\ 0&1&0&0&0&0&0&1 \\ 0&1&0&1&1&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&1&1&0&0&0&0 \\ 0&0&0&0&0&0&1&0 \\ 1&0&0&1&0&0&0&0 \\ 0&1&0&0&0&1&0&0 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}}$$

ここで，$x = x_0|x_1|x_2|x_3|x_4|x_5|x_6|x_7, y = y_0|y_1|y_2|y_3|y_4|y_5|y_6|y_7, x_i, y_i \in \{0,1\}$ とする．f と g の定数を 16 進数で表現すると，それぞれ `0x1e` と `0x69` である．

2.2.3 拡散行列

F 関数で用いられる 2 つの行列 M_0，M_1 を以下に定義する．

$$M_0 = \begin{pmatrix} \mathtt{0x01} & \mathtt{0x02} & \mathtt{0x04} & \mathtt{0x06} \\ \mathtt{0x02} & \mathtt{0x01} & \mathtt{0x06} & \mathtt{0x04} \\ \mathtt{0x04} & \mathtt{0x06} & \mathtt{0x01} & \mathtt{0x02} \\ \mathtt{0x06} & \mathtt{0x04} & \mathtt{0x02} & \mathtt{0x01} \end{pmatrix}, \quad M_1 = \begin{pmatrix} \mathtt{0x01} & \mathtt{0x08} & \mathtt{0x02} & \mathtt{0x0a} \\ \mathtt{0x08} & \mathtt{0x01} & \mathtt{0x0a} & \mathtt{0x02} \\ \mathtt{0x02} & \mathtt{0x0a} & \mathtt{0x01} & \mathtt{0x08} \\ \mathtt{0x0a} & \mathtt{0x02} & \mathtt{0x08} & \mathtt{0x01} \end{pmatrix} .$$

行列とベクトル間で実行される乗算は，辞書的順序で最初となる原始多項式 $z^8 + z^4 + z^3 + z^2 + 1$ で定義される GF(2^8) 上の演算として実行される．

2.3 データ処理部

2.3.1 全体構造

CLEFIA のデータ処理部は，暗号化関数 ENC_r，復号関数 DEC_r から構成される．ENC_r，DEC_r はそれぞれ 4 系列一般化 Feistel 構造 $GFN_{4,r}$, $GFN_{4,r}^{-1}$ に基づいている．P，C をそれぞれ 128 ビットの平文，暗号文とし，$P_i, C_i \in \{0,1\}^{32}$ $(0 \leq i < 4)$ を分割した平文，暗号文とする．但し，$P = P_0|P_1|P_2|P_3$, $C = C_0|C_1|C_2|C_3$ である．また $WK_0, WK_1, WK_2, WK_3 \in \{0,1\}^{32}$ をホワイトニング鍵，$RK_i \in \{0,1\}^{32}$ $(0 \leq i < 2r)$ を鍵スケジューリング部から出力されるラウンド鍵とする．このとき r ラウンドの暗号化関数 ENC_r は以下のように定義される．

$$ENC_r : \begin{cases} \{\{0,1\}^{32}\}^4 \times \{\{0,1\}^{32}\}^{2r} \times \{\{0,1\}^{32}\}^4 \to \{\{0,1\}^{32}\}^4 \\ (WK_{0(32)}, \ldots, WK_{3(32)}, RK_{0(32)}, \ldots, RK_{2r-1(32)}, P_{0(32)}, \ldots, P_{3(32)}) \\ \qquad\qquad \mapsto C_{0(32)}, \ldots, C_{3(32)} \end{cases}$$

Step 1.	$T_0 \mid T_1 \mid T_2 \mid T_3 \leftarrow P_0 \mid (P_1 \oplus WK_0) \mid P_2 \mid (P_3 \oplus WK_1)$
Step 2.	$T_0 \mid T_1 \mid T_2 \mid T_3 \leftarrow GFN_{4,r}(RK_0, \ldots, RK_{2r-1}, T_0, T_1, T_2, T_3)$
Step 3.	$C_0 \mid C_1 \mid C_2 \mid C_3 \leftarrow T_0 \mid (T_1 \oplus WK_2) \mid T_2 \mid (T_3 \oplus WK_3)$

復号関数 DEC_r は以下のように定義される．

$$DEC_r : \begin{cases} \{\{0,1\}^{32}\}^4 \times \{\{0,1\}^{32}\}^{2r} \times \{\{0,1\}^{32}\}^4 \to \{\{0,1\}^{32}\}^4 \\ (WK_{0(32)}, \ldots, WK_{3(32)}, RK_{0(32)}, \ldots, RK_{2r-1(32)}, C_{0(32)}, \ldots, C_{3(32)}) \\ \qquad\qquad \mapsto P_{0(32)}, \ldots, P_{3(32)} \end{cases}$$

Step 1.	$T_0 \mid T_1 \mid T_2 \mid T_3 \leftarrow C_0 \mid (C_1 \oplus WK_2) \mid C_2 \mid (C_3 \oplus WK_3)$
Step 2.	$T_0 \mid T_1 \mid T_2 \mid T_3 \leftarrow GFN_{4,r}^{-1}(RK_0, \ldots, RK_{2r-1}, T_0, T_1, T_2, T_3)$
Step 3.	$P_0 \mid P_1 \mid P_2 \mid P_3 \leftarrow T_0 \mid (T_1 \oplus WK_0) \mid T_2 \mid (T_3 \oplus WK_1)$

ENC_r 及び DEC_r の構造を図 2.3 に示す．

2.3.2 ラウンド数

ラウンド数 r は 128 ビット鍵の場合は 18, 192 ビットの場合は 22, 256 ビットの場合では 26 となる．また，ラウンド鍵 RK_i も鍵長によって異なり，128 ビット鍵の場合は 36, 192 ビットの場合は 44, 256 ビットの場合では 56 となる．

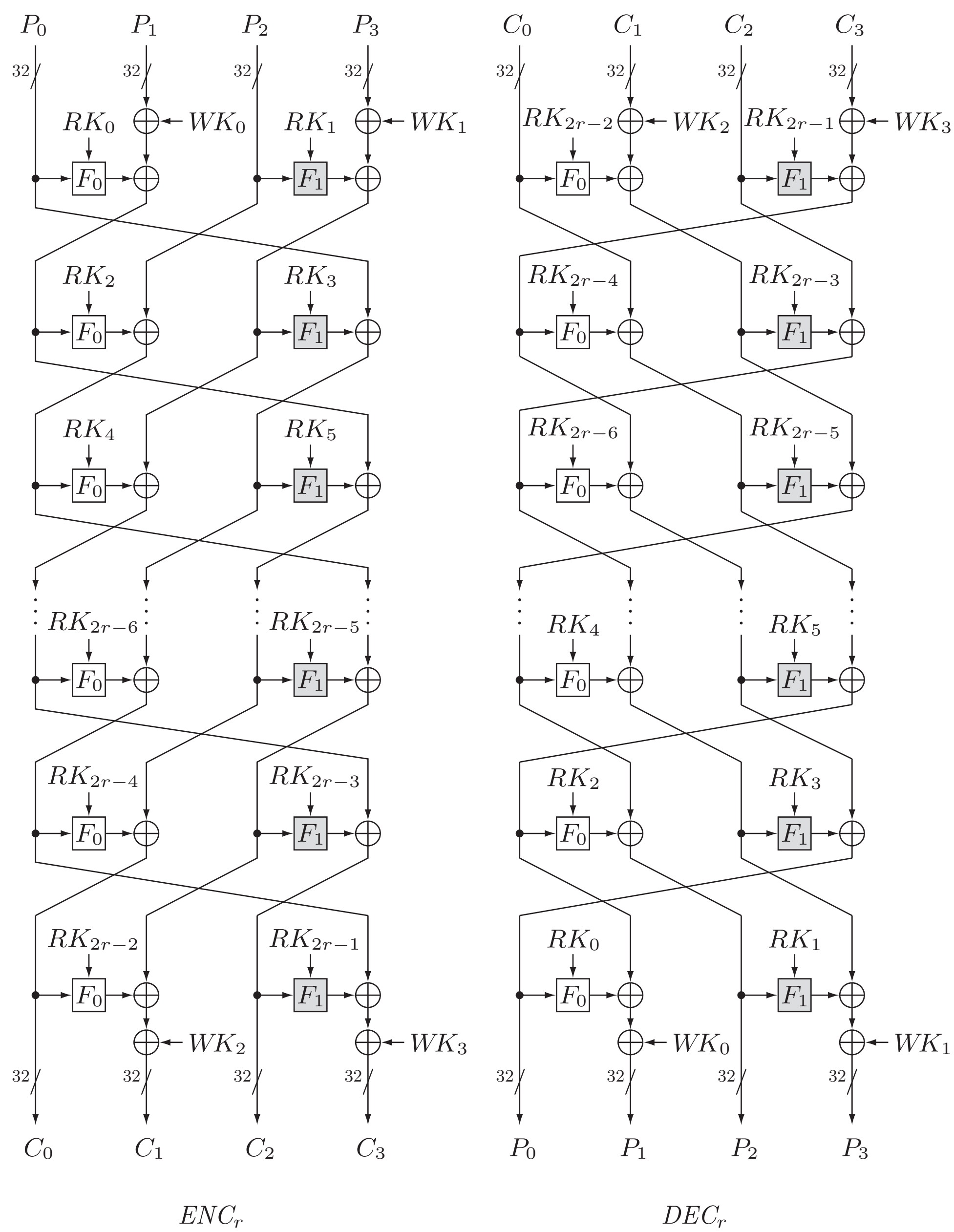

図 2.3: データ処理部の構造

2.4　鍵スケジューリング部

CLEFIA の鍵スケジューリング部は，128, 192 及び 256 ビット鍵を入力とし，ホワイトニング鍵 WK_i $(0 \leq i < 4)$ 及びラウンド鍵 RK_j $(0 \leq j < 2r)$ をデータ処理部に対して出力する．まずはじめに鍵スケジューリング部で用いられる $DoubleSwap$ 関数を定義する．

定義 2.1 $DoubleSwap$ 関数 Σ
$DoubleSwap$ 関数 $\Sigma : \{0,1\}^{128} \rightarrow \{0,1\}^{128}$ を以下に定義する．

$$X_{(128)} \mapsto Y_{(128)}$$
$$Y = X[7-63] \mid X[121-127] \mid X[0-6] \mid X[64-120] \ ,$$

$X[a-b]$ は X の a ビット目から b ビット目を切り出したデータを表す．但し，0 ビット目を最上位ビット *(MSB)* とする．

$DoubleSwap$ 関数を図 2.4 に図示する．

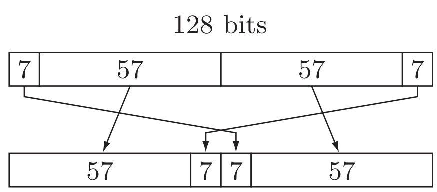

図 2.4: $DoubleSwap$ 関数 Σ

2.4.1　全体構造

鍵スケジューリング部はデータ処理部に対してホワイトニング鍵とラウンド鍵を出力する．K を秘密鍵，L を中間鍵とすると，鍵スケジュール部は以下の 2 つのステップより構成される．

1. 秘密鍵 K から中間鍵 L を生成
2. K と L を拡大し，ホワイトニング鍵 WK_i とラウンド鍵 RK_j を生成

秘密鍵 K から中間鍵 L を生成するために，128 ビット鍵の場合，128 ビット置換である $GFN_{4,12}$ を，192，256 鍵の場合，256 ビット置換である $GFN_{8,10}$ を用いる．

2.4.2 128 ビット鍵の鍵スケジューリング部

128 ビット中間鍵 L は，入力として $K = K_0|K_1|K_2|K_3$，ラウンド鍵として 24 個の 32 ビット定数 $CON_i^{(128)}$ $(0 \leq i < 24)$ を設定した場合の $GFN_{4,12}$ の出力として表現される．次にこれら秘密鍵 K，中間鍵 L を使い，ホワイトニング鍵 WK_i $(0 \leq i < 4)$ とラウンド鍵 RK_j $(0 \leq j < 36)$ を以下の手順で生成する．ここでは 36 個の 32 ビット定数 $CON_i^{(128)}$ $(24 \leq i < 60)$ を使用する．これら $CON_i^{(128)}$ の生成法については節 2.4.5 にて説明する．

(K から L の生成)

Step 1. $L \leftarrow GFN_{4,12}(CON_0^{(128)}, \ldots, CON_{23}^{(128)}, K_0, \ldots, K_3)$

(K と L の拡大)

Step 2. $WK_0|WK_1|WK_2|WK_3 \leftarrow K$

Step 3. $i = 0$ から 8 に対して以下を実行:

  $T \leftarrow L \oplus (CON_{24+4i}^{(128)} \mid CON_{24+4i+1}^{(128)} \mid CON_{24+4i+2}^{(128)} \mid CON_{24+4i+3}^{(128)})$

  $L \leftarrow \Sigma(L)$

  もし i が奇数ならば　$T \leftarrow T \oplus K$

  $RK_{4i}|RK_{4i+1}|RK_{4i+2}|RK_{4i+3} \leftarrow T$

図 2.5 はラウンド鍵と生成に必要なデータとの関係を示している．

WK_0	WK_1	WK_2	WK_3	$\leftarrow K$
RK_0	RK_1	RK_2	RK_3	$\leftarrow L \oplus (CON_{24}^{(128)}\|CON_{25}^{(128)}\|CON_{26}^{(128)}\|CON_{27}^{(128)})$
RK_4	RK_5	RK_6	RK_7	$\leftarrow \Sigma(L) \oplus K \oplus (CON_{28}^{(128)}\|CON_{29}^{(128)}\|CON_{30}^{(128)}\|CON_{31}^{(128)})$
RK_8	RK_9	RK_{10}	RK_{11}	$\leftarrow \Sigma^2(L) \oplus (CON_{32}^{(128)}\|CON_{33}^{(128)}\|CON_{34}^{(128)}\|CON_{35}^{(128)})$
RK_{12}	RK_{13}	RK_{14}	RK_{15}	$\leftarrow \Sigma^3(L) \oplus K \oplus (CON_{36}^{(128)}\|CON_{37}^{(128)}\|CON_{38}^{(128)}\|CON_{39}^{(128)})$
RK_{16}	RK_{17}	RK_{18}	RK_{19}	$\leftarrow \Sigma^4(L) \oplus (CON_{40}^{(128)}\|CON_{41}^{(128)}\|CON_{42}^{(128)}\|CON_{43}^{(128)})$
RK_{20}	RK_{21}	RK_{22}	RK_{23}	$\leftarrow \Sigma^5(L) \oplus K \oplus (CON_{44}^{(128)}\|CON_{45}^{(128)}\|CON_{46}^{(128)}\|CON_{47}^{(128)})$
RK_{24}	RK_{25}	RK_{26}	RK_{27}	$\leftarrow \Sigma^6(L) \oplus (CON_{48}^{(128)}\|CON_{49}^{(128)}\|CON_{50}^{(128)}\|CON_{51}^{(128)})$
RK_{28}	RK_{29}	RK_{30}	RK_{31}	$\leftarrow \Sigma^7(L) \oplus K \oplus (CON_{52}^{(128)}\|CON_{53}^{(128)}\|CON_{54}^{(128)}\|CON_{55}^{(128)})$
RK_{32}	RK_{33}	RK_{34}	RK_{35}	$\leftarrow \Sigma^8(L) \oplus (CON_{56}^{(128)}\|CON_{57}^{(128)}\|CON_{58}^{(128)}\|CON_{59}^{(128)})$

図 2.5: K と L の拡大 (128 ビット鍵)

2.4.3 192 ビット鍵の鍵スケジューリング部

2 つの 128 ビット鍵 K_L, K_R は 192 ビット秘密鍵 $K = K_0|K_1|K_2\,|K_3|K_4|K_5$, $K_i \in \{0,1\}^{32}$ より生成される．また 2 つの 128 ビット中間鍵 L_L, L_R は $K_L|K_R$ を 256 ビット入力，$CON_i^{(192)}$ $(0 \leq i < 40)$ をラウンド鍵として設

定した場合の $GFN_{8,10}$ の出力として表現される．図 2.6 に $GFN_{8,10}$ の構造を示す．

次にこれら鍵スケジュール入力鍵 K_L, K_R，中間鍵 L_L, L_R を使い，ホワイトニング鍵 WK_i $(0 \leq i < 4)$ とラウンド鍵 RK_j $(0 \leq j < 44)$ は以下の手順にて生成される．ここでは 44 個の 32 ビット定数 $CON_i^{(192)}$ を使用する．以下に 192 ビット/256 ビット鍵スケジューリング部の手順を示す．192 ビット鍵の場合，この手順において k を 192 に設定する．

(k ビット鍵での K_L, K_R から L_L, L_R の生成)
Step 1. $k = 192$ もしくは $k = 256$ を設定する
Step 2. $k = 192$ の場合： $K_L \leftarrow K_0|K_1|K_2|K_3$，$K_R \leftarrow K_4|K_5|\overline{K_0}|\overline{K_1}$
$k = 256$ の場合： $K_L \leftarrow K_0|K_1|K_2|K_3$，$K_R \leftarrow K_4|K_5|K_6|K_7$
Step 3. $K_L = K_{L0}|K_{L1}|K_{L2}|K_{L3}$，$K_R = K_{R0}|K_{R1}|K_{R2}|K_{R3}$ とする
$L_L|L_R \leftarrow$
$GFN_{8,10}(CON_0^{(k)}, \ldots, CON_{39}^{(k)}, K_{L0}, \ldots, K_{L3}, K_{R0}, \ldots, K_{R3})$

(k ビット鍵での K_L, K_R と L_L, L_R の拡大)
Step 4. $WK_0|WK_1|WK_2|WK_3 \leftarrow K_L \oplus K_R$
Step 5. $i = 0$ から 10 ($k = 192$ の場合) もしくは 12 ($k = 256$ の場合)
に対して以下を実行:
もし $(i \bmod 4) = 0$ または 1 ならば
$T \leftarrow L_L \oplus (CON_{40+4i}^{(k)} \mid CON_{40+4i+1}^{(k)} \mid CON_{40+4i+2}^{(k)} \mid CON_{40+4i+3}^{(k)})$
$L_L \leftarrow \Sigma(L_L)$
i が奇数ならば　$T \leftarrow T \oplus K_R$
それ以外ならば
$T \leftarrow L_R \oplus (CON_{40+4i}^{(k)} \mid CON_{40+4i+1}^{(k)} \mid CON_{40+4i+2}^{(k)} \mid CON_{40+4i+3}^{(k)})$
$L_R \leftarrow \Sigma(L_R)$
i が奇数ならば　$T \leftarrow T \oplus K_L$
$RK_{4i}|RK_{4i+1}|RK_{4i+2}|RK_{4i+3} \leftarrow T$

図 2.7 は各ラウンド鍵の生成に必要とされるデータとの関係を示している．

2.4.4　256 ビット鍵の鍵スケジューリング部

256 ビット鍵における鍵スケジューリング部は 192 ビット鍵におけるものと定数やラウンド鍵 RK_i の数，K_R の初期化を除いてほとんど同じである．

256 ビット鍵の場合は k を 256 にセットし，192 ビット鍵の場合とほぼ同じような処理手順を行う．中間鍵 L_L，L_R を生成するために，定数 $CON_i^{(256)}$ $(0 \leq i < 40)$ を用い，ラウンド鍵 RK_j $(0 \leq j < 52)$ を生成するために 52 個の 32 ビット定数 $CON_i^{(256)}$ $(40 \leq i < 92)$ を用いる．

図 2.8 は各ラウンド鍵と生成に必要とされるデータとの関係を示している．

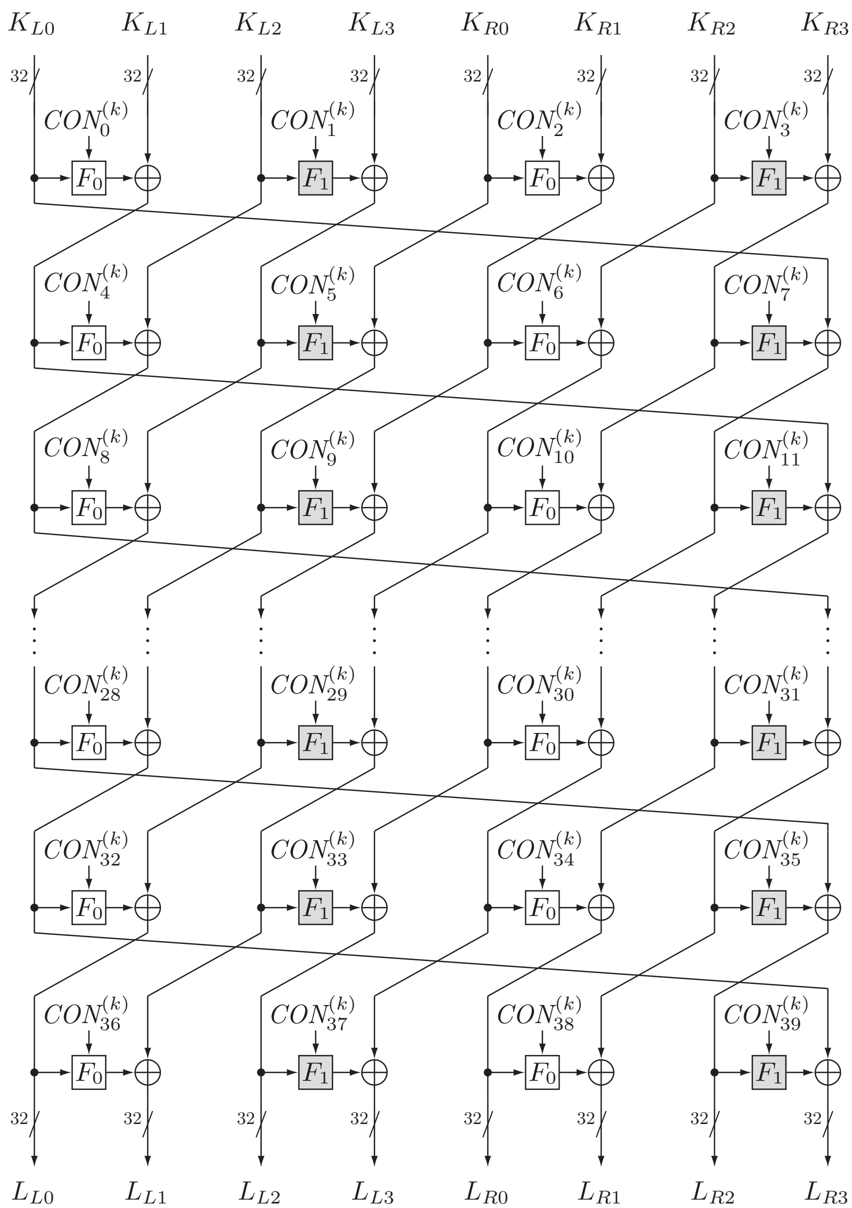

図 2.6: $GFN_{8,10}$ の構造

WK_0	WK_1	WK_2	WK_3	$\leftarrow K_L \oplus K_R$
RK_0	RK_1	RK_2	RK_3	$\leftarrow L_L \oplus (CON^{(192)}_{40} \mid CON^{(192)}_{41} \mid CON^{(192)}_{42} \mid CON^{(192)}_{43})$
RK_4	RK_5	RK_6	RK_7	$\leftarrow \Sigma(L_L) \oplus K_R \oplus (CON^{(192)}_{44} \mid CON^{(192)}_{45} \mid CON^{(192)}_{46} \mid CON^{(192)}_{47})$
RK_8	RK_9	RK_{10}	RK_{11}	$\leftarrow L_R \oplus (CON^{(192)}_{48} \mid CON^{(192)}_{49} \mid CON^{(192)}_{50} \mid CON^{(192)}_{51})$
RK_{12}	RK_{13}	RK_{14}	RK_{15}	$\leftarrow \Sigma(L_R) \oplus K_L \oplus (CON^{(192)}_{52} \mid CON^{(192)}_{53} \mid CON^{(192)}_{54} \mid CON^{(192)}_{55})$
RK_{16}	RK_{17}	RK_{18}	RK_{19}	$\leftarrow \Sigma^2(L_L) \oplus (CON^{(192)}_{56} \mid CON^{(192)}_{57} \mid CON^{(192)}_{58} \mid CON^{(192)}_{59})$
RK_{20}	RK_{21}	RK_{22}	RK_{23}	$\leftarrow \Sigma^3(L_L) \oplus K_R \oplus (CON^{(192)}_{60} \mid CON^{(192)}_{61} \mid CON^{(192)}_{62} \mid CON^{(192)}_{63})$
RK_{24}	RK_{25}	RK_{26}	RK_{27}	$\leftarrow \Sigma^2(L_R) \oplus (CON^{(192)}_{64} \mid CON^{(192)}_{65} \mid CON^{(192)}_{66} \mid CON^{(192)}_{67})$
RK_{28}	RK_{29}	RK_{30}	RK_{31}	$\leftarrow \Sigma^3(L_R) \oplus K_L \oplus (CON^{(192)}_{68} \mid CON^{(192)}_{69} \mid CON^{(192)}_{70} \mid CON^{(192)}_{71})$
RK_{32}	RK_{33}	RK_{34}	RK_{35}	$\leftarrow \Sigma^4(L_L) \oplus (CON^{(192)}_{72} \mid CON^{(192)}_{73} \mid CON^{(192)}_{74} \mid CON^{(192)}_{75})$
RK_{36}	RK_{37}	RK_{38}	RK_{39}	$\leftarrow \Sigma^5(L_L) \oplus K_R \oplus (CON^{(192)}_{76} \mid CON^{(192)}_{77} \mid CON^{(192)}_{78} \mid CON^{(192)}_{79})$
RK_{40}	RK_{41}	RK_{42}	RK_{43}	$\leftarrow \Sigma^4(L_R) \oplus (CON^{(192)}_{80} \mid CON^{(192)}_{81} \mid CON^{(192)}_{82} \mid CON^{(192)}_{83})$

図 2.7: K_L, K_R, L_L, L_R の拡大 (192-bit key)

WK_0	WK_1	WK_2	WK_3	$\leftarrow K_L \oplus K_R$
RK_0	RK_1	RK_2	RK_3	$\leftarrow L_L \oplus (CON^{(256)}_{40} \mid CON^{(256)}_{41} \mid CON^{(256)}_{42} \mid CON^{(256)}_{43})$
RK_4	RK_5	RK_6	RK_7	$\leftarrow \Sigma(L_L) \oplus K_R \oplus (CON^{(256)}_{44} \mid CON^{(256)}_{45} \mid CON^{(256)}_{46} \mid CON^{(256)}_{47})$
RK_8	RK_9	RK_{10}	RK_{11}	$\leftarrow L_R \oplus (CON^{(256)}_{48} \mid CON^{(256)}_{49} \mid CON^{(256)}_{50} \mid CON^{(256)}_{51})$
RK_{12}	RK_{13}	RK_{14}	RK_{15}	$\leftarrow \Sigma(L_R) \oplus K_L \oplus (CON^{(256)}_{52} \mid CON^{(256)}_{53} \mid CON^{(256)}_{54} \mid CON^{(256)}_{55})$
RK_{16}	RK_{17}	RK_{18}	RK_{19}	$\leftarrow \Sigma^2(L_L) \oplus (CON^{(256)}_{56} \mid CON^{(256)}_{57} \mid CON^{(256)}_{58} \mid CON^{(256)}_{59})$
RK_{20}	RK_{21}	RK_{22}	RK_{23}	$\leftarrow \Sigma^3(L_L) \oplus K_R \oplus (CON^{(256)}_{60} \mid CON^{(256)}_{61} \mid CON^{(256)}_{62} \mid CON^{(256)}_{63})$
RK_{24}	RK_{25}	RK_{26}	RK_{27}	$\leftarrow \Sigma^2(L_R) \oplus (CON^{(256)}_{64} \mid CON^{(256)}_{65} \mid CON^{(256)}_{66} \mid CON^{(256)}_{67})$
RK_{28}	RK_{29}	RK_{30}	RK_{31}	$\leftarrow \Sigma^3(L_R) \oplus K_L \oplus (CON^{(256)}_{68} \mid CON^{(256)}_{69} \mid CON^{(256)}_{70} \mid CON^{(256)}_{71})$
RK_{32}	RK_{33}	RK_{34}	RK_{35}	$\leftarrow \Sigma^4(L_L) \oplus (CON^{(256)}_{72} \mid CON^{(256)}_{73} \mid CON^{(256)}_{74} \mid CON^{(256)}_{75})$
RK_{36}	RK_{37}	RK_{38}	RK_{39}	$\leftarrow \Sigma^5(L_L) \oplus K_R \oplus (CON^{(256)}_{76} \mid CON^{(256)}_{77} \mid CON^{(256)}_{78} \mid CON^{(256)}_{79})$
RK_{40}	RK_{41}	RK_{42}	RK_{43}	$\leftarrow \Sigma^4(L_R) \oplus (CON^{(256)}_{80} \mid CON^{(256)}_{81} \mid CON^{(256)}_{82} \mid CON^{(256)}_{83})$
RK_{44}	RK_{45}	RK_{46}	RK_{47}	$\leftarrow \Sigma^5(L_R) \oplus K_L \oplus (CON^{(256)}_{84} \mid CON^{(256)}_{85} \mid CON^{(256)}_{86} \mid CON^{(256)}_{87})$
RK_{48}	RK_{49}	RK_{50}	RK_{51}	$\leftarrow \Sigma^6(L_L) \oplus (CON^{(256)}_{88} \mid CON^{(256)}_{89} \mid CON^{(256)}_{90} \mid CON^{(256)}_{91})$

図 2.8: K_L, K_R, L_L, L_R の拡大 (256-bit key)

2.4.5 定数

鍵スケジュール部では，32 ビット定数 $CON_i^{(k)}$ が用いられている．鍵長に応じてそれぞれ，128 ビット鍵のとき 60 個，192 ビット鍵のとき 84 個，256 ビット鍵では 92 個の定数が使用される．$\mathbf{P}_{(16)}$ = `0xb7e1` ($= (e-2)\cdot 2^{16}$) (e は自然対数 (2.71828...))，$\mathbf{Q}_{(16)}$ = `0x243f` ($= (\pi - 3)\cdot 2^{16}$) (π は円周率 (3.14159...)) とすると，定数 $CON_i^{(k)} (k = 128, 192, 256)$ は，以下の手順で生成される．ここで繰り返し回数 $l^{(k)}$ 及び初期値 $IV^{(k)}$ は表 2.4 に定義された値である．

Step 1. $T_0 \leftarrow IV^{(k)}$
Step 2. $i = 0$ から $l^{(k)} - 1$ に対して以下を実行：
  Step 2.1. $CON_{2i}^{(k)} \leftarrow (T_i \oplus \mathbf{P}) \mid (\overline{T_i} \lll 1)$
  Step 2.2. $CON_{2i+1}^{(k)} \leftarrow (\overline{T_i} \oplus \mathbf{Q}) \mid (T_i \lll 8)$
  Step 2.3. $T_{i+1} \leftarrow T_i \cdot \mathtt{0x0002}^{-1}$

また Step 2.3 における乗算は原始多項式 $z^{16}+z^{15}+z^{13}+z^{11}+z^5+z^4+1$ (=`0x1a831`)[1]で定義される GF(2^{16}) 上の演算として実行される．

表 2.4: CON の個数および IV

k	# of $CON_i^{(k)}$	$l^{(k)}$	$IV^{(k)}$	
128	60	30	`0x428a`	$(= (\sqrt[3]{2}-1)\cdot 2^{16})$
192	84	42	`0x7137`	$(= (\sqrt[3]{3}-1)\cdot 2^{16})$
256	92	46	`0xb5c0`	$(= (\sqrt[3]{5}-1)\cdot 2^{16})$

表 2.5 から表 2.7 は T_i の値，表 2.8 から表 2.12 は $CON_i^{(k)}$ の値 (16 進数表現) を示している．

[1] 下位 16 ビットの値は `0xa831`=$(\sqrt[3]{101}-4)\cdot 2^{16}$ と定義される，‘101’ は原始多項式となる最小の素数である．

表 2.5: $T_i^{(128)}$

i	0	1	2	3	4	5	6	7
$T_i^{(128)}$	428a	2145	c4ba	625d	e536	729b	ed55	a2b2
i	8	9	10	11	12	13	14	15
$T_i^{(128)}$	5159	fcb4	7e5a	3f2d	cb8e	65c7	e6fb	a765
i	16	17	18	19	20	21	22	23
$T_i^{(128)}$	87aa	43d5	f5f2	7af9	e964	74b2	3a59	c934
i	24	25	26	27	28	29		
$T_i^{(128)}$	649a	324d	cd3e	669f	e757	a7b3		

表 2.6: $T_i^{(192)}$

i	0	1	2	3	4	5	6	7
$T_i^{(192)}$	7137	ec83	a259	8534	429a	214d	c4be	625f
i	8	9	10	11	12	13	14	15
$T_i^{(192)}$	e537	a683	8759	97b4	4bda	25ed	c6ee	6377
i	16	17	18	19	20	21	22	23
$T_i^{(192)}$	e5a3	a6c9	877c	43be	21df	c4f7	b663	8f29
i	24	25	26	27	28	29	30	31
$T_i^{(192)}$	938c	49c6	24e3	c669	b72c	5b96	2dcb	c2fd
i	32	33	34	35	36	37	38	39
$T_i^{(192)}$	b566	5ab3	f941	a8b8	545c	2a2e	1517	de93
i	40	41						
$T_i^{(192)}$	bb51	89b0						

表 2.7: $T_i^{(256)}$

i	0	1	2	3	4	5	6	7
$T_i^{(256)}$	b5c0	5ae0	2d70	16b8	0b5c	05ae	02d7	d573
i	8	9	10	11	12	13	14	15
$T_i^{(256)}$	bea1	8b48	45a4	22d2	1169	dcac	6e56	372b
i	16	17	18	19	20	21	22	23
$T_i^{(256)}$	cf8d	b3de	59ef	f8ef	a86f	802f	940f	9e1f
i	24	25	26	27	28	29	30	31
$T_i^{(256)}$	9b17	9993	98d1	9870	4c38	261c	130e	0987
i	32	33	34	35	36	37	38	39
$T_i^{(256)}$	d0db	bc75	8a22	4511	f690	7b48	3da4	1ed2
i	40	41	42	43	44	45		
$T_i^{(256)}$	0f69	d3ac	69d6	34eb	ce6d	b32e		

表 2.8: $CON_i^{(128)}$ $(0 \leq i < 60)$

i	0	1	2	3
$CON_i^{(128)}$	f56b7aeb	994a8a42	96a4bd75	fa854521
i	4	5	6	7
$CON_i^{(128)}$	735b768a	1f7abac4	d5bc3b45	b99d5d62
i	8	9	10	11
$CON_i^{(128)}$	52d73592	3ef636e5	c57a1ac9	a95b9b72
i	12	13	14	15
$CON_i^{(128)}$	5ab42554	369555ed	1553ba9a	7972b2a2
i	16	17	18	19
$CON_i^{(128)}$	e6b85d4d	8a995951	4b550696	2774b4fc
i	20	21	22	23
$CON_i^{(128)}$	c9bb034b	a59a5a7e	88cc81a5	e4ed2d3f
i	24	25	26	27
$CON_i^{(128)}$	7c6f68e2	104e8ecb	d2263471	be07c765
i	28	29	30	31
$CON_i^{(128)}$	511a3208	3d3bfbe6	1084b134	7ca565a7
i	32	33	34	35
$CON_i^{(128)}$	304bf0aa	5c6aaa87	f4347855	9815d543
i	36	37	38	39
$CON_i^{(128)}$	4213141a	2e32f2f5	cd180a0d	a139f97a
i	40	41	42	43
$CON_i^{(128)}$	5e852d36	32a464e9	c353169b	af72b274
i	44	45	46	47
$CON_i^{(128)}$	8db88b4d	e199593a	7ed56d96	12f434c9
i	48	49	50	51
$CON_i^{(128)}$	d37b36cb	bf5a9a64	85ac9b65	e98d4d32
i	52	53	54	55
$CON_i^{(128)}$	7adf6582	16fe3ecd	d17e32c1	bd5f9f66
i	56	57	58	59
$CON_i^{(128)}$	50b63150	3c9757e7	1052b098	7c73b3a7

表 2.9: $CON_i^{(192)}$ $(0 \leq i < 60)$

i	0	1	2	3
$CON_i^{(192)}$	c6d61d91	aaf73771	5b6226f8	374383ec
i	4	5	6	7
$CON_i^{(192)}$	15b8bb4c	799959a2	32d5f596	5ef43485
i	8	9	10	11
$CON_i^{(192)}$	f57b7acb	995a9a42	96acbd65	fa8d4d21
i	12	13	14	15
$CON_i^{(192)}$	735f7682	1f7ebec4	d5be3b41	b99f5f62
i	16	17	18	19
$CON_i^{(192)}$	52d63590	3ef737e5	1162b2f8	7d4383a6
i	20	21	22	23
$CON_i^{(192)}$	30b8f14c	5c995987	2055d096	4c74b497
i	24	25	26	27
$CON_i^{(192)}$	fc3b684b	901ada4b	920cb425	fe2ded25
i	28	29	30	31
$CON_i^{(192)}$	710f7222	1d2eeec6	d4963911	b8b77763
i	32	33	34	35
$CON_i^{(192)}$	524234b8	3e63a3e5	1128b26c	7d09c9a6
i	36	37	38	39
$CON_i^{(192)}$	309df106	5cbc7c87	f45f7883	987ebe43
i	40	41	42	43
$CON_i^{(192)}$	963ebc41	fa1fdf21	73167610	1f37f7c4
i	44	45	46	47
$CON_i^{(192)}$	01829338	6da363b6	38c8e1ac	54e9298f
i	48	49	50	51
$CON_i^{(192)}$	246dd8e6	484c8c93	fe276c73	9206c649
i	52	53	54	55
$CON_i^{(192)}$	9302b639	ff23e324	7188732c	1da969c6
i	56	57	58	59
$CON_i^{(192)}$	00cd91a6	6cec2cb7	ec7748d3	8056965b

表 2.10: $CON_i^{(192)}$ $(60 \leq i < 84)$

i	60	61	62	63
$CON_i^{(192)}$	9a2aa469	f60bcb2d	751c7a04	193dfdc2
i	64	65	66	67
$CON_i^{(192)}$	02879532	6ea666b5	ed524a99	8173b35a
i	68	69	70	71
$CON_i^{(192)}$	4ea00d7c	228141f9	1f59ae8e	7378b8a8
i	72	73	74	75
$CON_i^{(192)}$	e3bd5747	8f9c5c54	9dcfaba3	f1ee2e2a
i	76	77	78	79
$CON_i^{(192)}$	a2f6d5d1	ced71715	697242d8	055393de
i	80	81	82	83
$CON_i^{(192)}$	0cb0895c	609151bb	3e51ec9e	5270b089

表 2.11: $CON_i^{(256)}$ $(0 \leq i < 24)$

i	0	1	2	3
$CON_i^{(256)}$	0221947e	6e00c0b5	ed014a3f	8120e05a
i	4	5	6	7
$CON_i^{(256)}$	9a91a51f	f6b0702d	a159d28f	cd78b816
i	8	9	10	11
$CON_i^{(256)}$	bcbde947	d09c5c0b	b24ff4a3	de6eae05
i	12	13	14	15
$CON_i^{(256)}$	b536fa51	d917d702	62925518	0eb373d5
i	16	17	18	19
$CON_i^{(256)}$	094082bc	6561a1be	3ca9e96e	5088488b
i	20	21	22	23
$CON_i^{(256)}$	f24574b7	9e64a445	9533ba5b	f912d222

表 2.12: $CON_i^{(256)}$ $(24 \le i < 92)$

i	24	25	26	27
$CON_i^{(256)}$	a688dd2d	caa96911	6b4d46a6	076cacdc
i	28	29	30	31
$CON_i^{(256)}$	d9b72353	b596566e	80ca91a9	eceb2b37
i	32	33	34	35
$CON_i^{(256)}$	786c60e4	144d8dcf	043f9842	681edeb3
i	36	37	38	39
$CON_i^{(256)}$	ee0e4c21	822fef59	4f0e0e20	232feff8
i	40	41	42	43
$CON_i^{(256)}$	1f8eaf20	73af6fa8	37ceffa0	5bef2f80
i	44	45	46	47
$CON_i^{(256)}$	23eed7e0	4fcf0f94	29fec3c0	45df1f9e
i	48	49	50	51
$CON_i^{(256)}$	2cf6c9d0	40d7179b	2e72ccd8	42539399
i	52	53	54	55
$CON_i^{(256)}$	2f30ce5c	4311d198	2f91cf1e	43b07098
i	56	57	58	59
$CON_i^{(256)}$	fbd9678f	97f8384c	91fdb3c7	fddc1c26
i	60	61	62	63
$CON_i^{(256)}$	a4efd9e3	c8ce0e13	be66ecf1	d2478709
i	64	65	66	67
$CON_i^{(256)}$	673a5e48	0b1bdbd0	0b948714	67b575bc
i	68	69	70	71
$CON_i^{(256)}$	3dc3ebba	51e2228a	f2f075dd	9ed11145
i	72	73	74	75
$CON_i^{(256)}$	417112de	2d5090f6	cca9096f	a088487b
i	76	77	78	79
$CON_i^{(256)}$	8a4584b7	e664a43d	a933c25b	c512d21e
i	80	81	82	83
$CON_i^{(256)}$	b888e12d	d4a9690f	644d58a6	086cacd3
i	84	85	86	87
$CON_i^{(256)}$	de372c53	b216d669	830a9629	ef2beb34
i	88	89	90	91
$CON_i^{(256)}$	798c6324	15ad6dce	04cf99a2	68ee2eb3

2.5 テストベクトル

各鍵長に対する CLEFIA のテストベクトルを示す．データは 16 進数表現である．

128-bit key:

key	`ffeeddcc bbaa9988 77665544 33221100`
plaintext	`00010203 04050607 08090a0b 0c0d0e0f`
ciphertext	`de2bf2fd 9b74aacd f1298555 459494fd`

192-bit key:

key	`ffeeddcc bbaa9988 77665544 33221100`
	`f0e0d0c0 b0a09080`
plaintext	`00010203 04050607 08090a0b 0c0d0e0f`
ciphertext	`e2482f64 9f028dc4 80dda184 fde181ad`

256-bit key:

key	`ffeeddcc bbaa9988 77665544 33221100`
	`f0e0d0c0 b0a09080 70605040 30201000`
plaintext	`00010203 04050607 08090a0b 0c0d0e0f`
ciphertext	`a1397814 289de80c 10da46d1 fa48b38a`

2.5.1 テストベクトル (中間値)

128-bit key:

key	ffeeddcc bbaa9988 77665544 33221100
plaintext	00010203 04050607 08090a0b 0c0d0e0f
ciphertext	de2bf2fd 9b74aacd f1298555 459494fd
L	8f89a61b 9db9d0f3 93e65627 da0d027e
$WK_{0,1,2,3}$	ffeeddcc bbaa9988 77665544 33221100
$RK_{0,1,2,3}$	f3e6cef9 8df75e38 41c06256 640ac51b
$RK_{4,5,6,7}$	6a27e20a 5a791b90 e8c528dc 00336ea3
$RK_{8,9,10,11}$	59cd17c4 28565583 312a37cc c08abd77
$RK_{12,13,14,15}$	7e8e7eec 8be7e949 d3f463d6 a0aad6aa
$RK_{16,17,18,19}$	e75eb039 0d657eb9 018002e2 9117d009
$RK_{20,21,22,23}$	9f98d11e babee8cf b0369efa d3aaef0d
$RK_{24,25,26,27}$	3438f93b f9cea4a0 68df9029 b869b4a7
$RK_{28,29,30,31}$	24d6406d e74bc550 41c28193 16de4795
$RK_{32,33,34,35}$	a34a20f5 33265d14 b19d0554 5142f434

plaintext		00010203	04050607	08090a0b	0c0d0e0f
initial whitening key			ffeeddcc		bbaa9988
after whitening		00010203	fbebdbcb	08090a0b	b7a79787
Round 1	input	00010203	fbebdbcb	08090a0b	b7a79787
	F-function	F_0		F_1	
	input	00010203		08090a0b	
	round key	f3e6cef9		8df75e38	
	after key add	f3e7ccfa		85fe5433	
	after S	290246e1		777de8e8	
	after M	547a3193		abf12070	
Round 2	input	af91ea58	08090a0b	1c56b7f7	00010203
	F-function	F_0		F_1	
	input	af91ea58		1c56b7f7	
	round key	41c06256		640ac51b	
	after key add	ee51880e		785c72ec	
	after S	cb5d2b0c		63a5edd2	
	after M	f51cebb3		82dfe347	
Round 3	input	fd15e1b8	1c56b7f7	82dee144	af91ea58
	F-function	F_0		F_1	
	input	fd15e1b8		82dee144	
	round key	6a27e20a		5a791b90	
	after key add	973203b2		d8a7fad4	
	after S	c2c7c6c2		be59e10d	
	after M	d8dfd8de		e15ea81c	
Round 4	input	c4896f29	82dee144	4ecf4244	fd15e1b8
	F-function	F_0		F_1	
	input	c4896f29		4ecf4244	
	round key	e8c528dc		00336ea3	
	after key add	2c4c47f5		4efc2ce7	
	after S	9da4dafc		43bce638	
	after M	b5b28e96		b65c519a	
Round 5	input	376c6fd2	4ecf4244	4b49b022	c4896f29
	F-function	F_0		F_1	
	input	376c6fd2		4b49b022	
	round key	59cd17c4		28565583	
	after key add	6ea17816		631fe5a1	
	after S	f26ad3e5		62af9f1b	
	after M	29f08afd		be01d127	
Round 6	input	673fc8b9	4b49b022	7a88be0e	376c6fd2
	F-function	F_0		F_1	
	input	673fc8b9		7a88be0e	
	round key	312a37cc		c08abd77	
	after key add	5615ff75		ba020379	
	after S	b39c8e58		2dd1e9a2	
	after M	5999a79e		0429b329	

Round 7	input	12d017bc	7a88be0e	3345dcfb	673fc8b9
	F-function	F_0		F_1	
	input	12d017bc		3345dcfb	
	round key	7e8e7eec		8be7e949	
	after key add	6c5e6950		b8a235b2	
	after S	8b737025		67a08eba	
	after M	6ed11b09		dfd3cd32	
Round 8	input	1459a507	3345dcfb	b8ec058b	12d017bc
	F-function	F_0		F_1	
	input	1459a507		b8ec058b	
	round key	d3f463d6		a0aad6aa	
	after key add	c7adc6d1		1846d321	
	after S	e7ee5a5f		9e97f1a1	
	after M	8c9d011c		93684eec	
Round 9	input	bfd8dde7	b8ec058b	81b85950	1459a507
	F-function	F_0		F_1	
	input	bfd8dde7		81b85950	
	round key	e75eb039		0d657eb9	
	after key add	58866dde		8cdd27e9	
	after S	4e821daf		59c56044	
	after M	e6d6501e		6d5839b4	
Round 10	input	5e3a5595	81b85950	79019cb3	bfd8dde7
	F-function	F_0		F_1	
	input	5e3a5595		79019cb3	
	round key	018002e2		9117d009	
	after key add	5fba5777		e8164cba	
	after S	612d8f7b		0185a49c	
	after M	3a1b0e97		b9b479c8	
Round 11	input	bba357c7	79019cb3	066ca42f	5e3a5595
	F-function	F_0		F_1	
	input	bba357c7		066ca42f	
	round key	9f98d11e		babee8cf	
	after key add	243b86d9		bcd24ce0	
	after S	f70f1144		cb72a481	
	after M	28974052		4a6700b1	
Round 12	input	5196dce1	066ca42f	145d5524	bba357c7
	F-function	F_0		F_1	
	input	5196dce1		145d5524	
	round key	b0369efa		d3aaef0d	
	after key add	e1a0421b		c7f7ba29	
	after S	6f7efd4f		72642dce	
	after M	ffb5db32		907d3820	

Round 13	input	f9d97f1d	145d5524	2bde6fe7	5196dce1
	F-function	F_0		F_1	
	input	f9d97f1d		2bde6fe7	
	round key	3438f93b		f9cea4a0	
	after key add	cde18626		d210cb47	
	after S	3f751141		ab28e0da	
	after M	0a744c28		1c3e38a3	
Round 14	input	1e29190c	2bde6fe7	4da8e442	f9d97f1d
	F-function	F_0		F_1	
	input	1e29190c		4da8e442	
	round key	68df9029		b869b4a7	
	after key add	76f68925		f5c150e5	
	after S	fe6db7e7		fc0c25f6	
	after M	aaa2c803		c4315b8d	
Round 15	input	817ca7e4	4da8e442	3de82490	1e29190c
	F-function	F_0		F_1	
	input	817ca7e4		3de82490	
	round key	24d6406d		e74bc550	
	after key add	a5aae789		daa3e1c0	
	after S	8d233818		2904757b	
	after M	7bd4cced		eac2f0fb	
Round 16	input	367c28af	3de82490	f4ebe9f7	817ca7e4
	F-function	F_0		F_1	
	input	367c28af		f4ebe9f7	
	round key	41c28193		16de4795	
	after key add	77bea93c		e235ae62	
	after S	7c4a935b		669b8953	
	after M	598e6940		c119609f	
Round 17	input	64664dd0	f4ebe9f7	4065c77b	367c28af
	F-function	F_0		F_1	
	input	64664dd0		4065c77b	
	round key	a34a20f5		33265d14	
	after key add	c72c6d25		73439a6f	
	after S	e7e61de7		788c85b4	
	after M	2ac01b0a		c755adfa	
Round 18	input	de2bf2fd	4065c77b	f1298555	64664dd0
	F-function	F_0		F_1	
	input	de2bf2fd		f1298555	
	round key	b19d0554		5142f434	
	after key add	6fb6f7a9		a06b7161	
	after S	b44d648c		7e99ea2a	
	after M	ac7738f2		12d0c82d	

output	de2bf2fd	ec12ff89	f1298555	76b685fd
final whitening key		77665544		33221100
after whitening	de2bf2fd	9b74aacd	f1298555	459494fd
ciphertext	de2bf2fd	9b74aacd	f1298555	459494fd

192-bit key:

key	`ffeeddcc bbaa9988 77665544 33221100`
	`f0e0d0c0 b0a09080`
plaintext	`00010203 04050607 08090a0b 0c0d0e0f`
ciphertext	`e2482f64 9f028dc4 80dda184 fde181ad`
L_L	`db05415a 800082db 7cb8186c d788c5f3`
L_R	`1ca9b2e1 b4606829 c92dd35e 2258a432`
$WK_{0,1,2,3}$	`0f0e0d0c 0b0a0908 77777777 77777777`
$RK_{0,1,2,3}$	`4d3bfd1b 7a1f5dfa 0fae6e7c c8bf3237`
$RK_{4,5,6,7}$	`73c2eeb8 dd429ec5 e220b3af c9135e73`
$RK_{8,9,10,11}$	`38c46a07 fc2ce4ba 370abf2d b05e627b`
$RK_{12,13,14,15}$	`38351b2f 74bd6e1e 1b7c7dce 92cfc98e`
$RK_{16,17,18,19}$	`509b31a6 4c5ad53c 6fc2ba33 e1e5c878`
$RK_{20,21,22,23}$	`419a74b9 1dd79e0e 240a33d2 9dabfd09`
$RK_{24,25,26,27}$	`6e3ff82a 74ac3ffd b9696e2e cc0b3a38`
$RK_{28,29,30,31}$	`ed785cbd 9c077c13 04978d83 2ec058ba`
$RK_{32,33,34,35}$	`4bbd5f6a 31fe8de8 b76da574 3a6fa8e7`
$RK_{36,37,38,39}$	`521213ce 4f1f59d8 c13624f6 ee91f6a4`
$RK_{40,41,42,43}$	`17f68fde f6c360a9 6288bc72 c0ad856b`

plaintext		00010203	04050607	08090a0b	0c0d0e0f
initial whitening key			0f0e0d0c		0b0a0908
after whitening		00010203	0b0b0b0b	08090a0b	07070707
Round 1	input	00010203	0b0b0b0b	08090a0b	07070707
	F-function	F_0		F_1	
	input	00010203		08090a0b	
	round key	4d3bfd1b		7a1f5dfa	
	after key add	4d3aff18		721657f1	
	after S	43c58e9e		ed85d736	
	after M	b5021a3b		c397f62b	
Round 2	input	be091130	08090a0b	c490f12c	00010203
	F-function	F_0		F_1	
	input	be091130		c490f12c	
	round key	0fae6e7c		c8bf3237	
	after key add	b1a77f4c		0c2fc31b	
	after S	f3d10ba4		13d83a3d	
	after M	9fba69c1		6683cae3	
Round 3	input	97b363ca	c490f12c	6682c8e0	be091130
	F-function	F_0		F_1	
	input	97b363ca		6682c8e0	
	round key	73c2eeb8		dd429ec5	
	after key add	e4718d72		bbc05625	
	after S	79ea66ed		f47b0d7a	
	after M	61c21ea5		120e06e2	
Round 4	input	a552ef89	6682c8e0	ac0717d2	97b363ca
	F-function	F_0		F_1	
	input	a552ef89		ac0717d2	
	round key	e220b3af		c9135e73	
	after key add	47725c26		651449a1	
	after S	daeda541		355c651b	
	after M	28a43c63		cb1ab573	
Round 5	input	4e26f483	ac0717d2	5ca9d6b9	a552ef89
	F-function	F_0		F_1	
	input	4e26f483		5ca9d6b9	
	round key	38c46a07		fc2ce4ba	
	after key add	76e29e84		a0853203	
	after S	fe663e39		7edcc7c6	
	after M	5ce7dafe		ac7f4e3e	
Round 6	input	f0e0cd2c	5ca9d6b9	092da1b7	4e26f483
	F-function	F_0		F_1	
	input	f0e0cd2c		092da1b7	
	round key	370abf2d		b05e627b	
	after key add	c7ea7201		b973c3cc	
	after S	e77f9fda		174a3a46	
	after M	b9869270		8fc7e089	

Round 7	input	e52f44c9 092da1b7	c1e1140a f0e0cd2c
	F-function	F_0	F_1
	input	e52f44c9	c1e1140a
	round key	38351b2f	74bd6e1e
	after key add	dd1a5fe6	b55c7a14
	after S	c5496150	5aa5c15c
	after M	33d8590f	e62eb913
Round 8	input	3af5f8b8 c1e1140a	16ce743f e52f44c9
	F-function	F_0	F_1
	input	3af5f8b8	16ce743f
	round key	1b7c7dce	92cfc98e
	after key add	21898576	8401bdb1
	after S	a118dc09	3949b1f3
	after M	f091202d	04f9e827
Round 9	input	31703427 16ce743f	e1d6acee 3af5f8b8
	F-function	F_0	F_1
	input	31703427	e1d6acee
	round key	509b31a6	4c5ad53c
	after key add	61eb0581	ad8c79d2
	after S	2a8d3304	eeffc072
	after M	f9639a90	8bebfe3d
Round 10	input	efadeeaf e1d6acee	b11e0685 31703427
	F-function	F_0	F_1
	input	efadeeaf	b11e0685
	round key	6fc2ba33	e1e5c878
	after key add	806f549c	50fbcefd
	after S	cd5eeb61	25d7fe02
	after M	a100e35b	26a4e16d
Round 11	input	40d64fb5 b11e0685	17d4d54a efadeeaf
	F-function	F_0	F_1
	input	40d64fb5	17d4d54a
	round key	419a74b9	1dd79e0e
	after key add	014c3b0c	0a034b44
	after S	49a4c013	b4c6c912
	after M	51c0208f	f1a2c339
Round 12	input	e0de260a 17d4d54a	1e0f2d96 40d64fb5
	F-function	F_0	F_1
	input	e0de260a	1e0f2d96
	round key	240a33d2	9dabfd09
	after key add	c4d415d8	83a4d09f
	after S	801beebe	86b8f8ed
	after M	8a9aef34	3e451646

Round 13	input	9d4e3a7e 1e0f2d96	7e9359f3 e0de260a
	F-function	F_0	F_1
	input	9d4e3a7e	7e9359f3
	round key	6e3ff82a	74ac3ffd
	after key add	f371c254	0a3f660e
	after S	29ea68e8	b4f530a8
	after M	17524741	4b8c607e
Round 14	input	095d6ad7 7e9359f3	ab524674 9d4e3a7e
	F-function	F_0	F_1
	input	095d6ad7	ab524674
	round key	b9696e2e	cc0b3a38
	after key add	b03404f9	67597c4c
	after S	152a2f03	52161e39
	after M	f7ee818b	7902f3eb
Round 15	input	897dd878 ab524674	e44cc995 095d6ad7
	F-function	F_0	F_1
	input	897dd878	e44cc995
	round key	ed785cbd	9c077c13
	after key add	640584c5	784bb586
	after S	459d9e10	636b5a11
	after M	4034defc	0228bdd4
Round 16	input	eb669888 e44cc995	0b75d703 897dd878
	F-function	F_0	F_1
	input	eb669888	0b75d703
	round key	04978d83	2ec058ba
	after key add	eff1150b	25b58fb9
	after S	90e4ee38	e7691f3b
	after M	4a678609	05b2b4a9
Round 17	input	ae2b4f9c 0b75d703	8ccf6cd1 eb669888
	F-function	F_0	F_1
	input	ae2b4f9c	8ccf6cd1
	round key	4bbd5f6a	31fe8de8
	after key add	e59610f6	bd31e139
	after S	f6a5286d	b15d7589
	after M	720df49d	bad65e22
Round 18	input	7978239e 8ccf6cd1	51b0c6aa ae2b4f9c
	F-function	F_0	F_1
	input	7978239e	51b0c6aa
	round key	b76da574	3a6fa8e7
	after key add	ce1586ea	6bdf6e4d
	after S	919c117f	283aaa43
	after M	ef24fe56	08916103

Round 19	input	63eb9287	51b0c6aa	a6ba2e9f	7978239e
	F-function	F_0		F_1	
	input	63eb9287		a6ba2e9f	
	round key	521213ce		4f1f59d8	
	after key add	31f98149		e9a57747	
	after S	5d03e265		3c8d7bda	
	after M	b7464b63		e1d086a7	
Round 20	input	e6f68dc9	a6ba2e9f	98a8a539	63eb9287
	F-function	F_0		F_1	
	input	e6f68dc9		98a8a539	
	round key	c13624f6		ee91f6a4	
	after key add	27c0a93f		7639539d	
	after S	20b5938b		09893194	
	after M	3cae819e		b603c454	
Round 21	input	9a14af01	98a8a539	d5e856d3	e6f68dc9
	F-function	F_0		F_1	
	input	9a14af01		d5e856d3	
	round key	17f68fde		f6c360a9	
	after key add	8de220df		232b367a	
	after S	6666bff2		b383a1bd	
	after M	7ae08a5d		662b2c4d	
Round 22	input	e2482f64	d5e856d3	80dda184	9a14af01
	F-function	F_0		F_1	
	input	e2482f64		80dda184	
	round key	6288bc72		c0ad856b	
	after key add	80c09316		407024ef	
	after S	cdb5f1e5		fbe99290	
	after M	3d9dac60		108259db	
output		e2482f64	e875fab3	80dda184	8a96f6da
final whitening key			77777777		77777777
after whitening		e2482f64	9f028dc4	80dda184	fde181ad
ciphertext		e2482f64	9f028dc4	80dda184	fde181ad

256-bit key:

key	ffeeddcc bbaa9988 77665544 33221100
	f0e0d0c0 b0a09080 70605040 30201000
plaintext	00010203 04050607 08090a0b 0c0d0e0f
ciphertext	a1397814 289de80c 10da46d1 fa48b38a
L_L	477e8f09 66ee5378 2cc2be04 bf55e28f
L_R	d6c10b89 4eeab575 84bd5663 cc933940
$WK_{0,1,2,3}$	0f0e0d0c 0b0a0908 07060504 03020100
$RK_{0,1,2,3}$	58f02029 15413cd0 1b0c41a4 e4bacd0f
$RK_{4,5,6,7}$	6c498393 8846231b 1fc716fc 7c81a45b
$RK_{8,9,10,11}$	fa37c259 0e3da2ee aacf9abb 8ec0aad9
$RK_{12,13,14,15}$	b05bd737 8de1f2d0 8ffee0f6 b70b47ea
$RK_{16,17,18,19}$	581b3e34 03263f89 2f7100cd 05cee171
$RK_{20,21,22,23}$	b523d4e9 176d7c44 6d7ba5d7 f797b2f3
$RK_{24,25,26,27}$	25d80df2 a646bba2 6a3a95e1 3e3a47f0
$RK_{28,29,30,31}$	b304eb20 44f8824e c7557cbc 47401e21
$RK_{32,33,34,35}$	d71ff7e9 aca1fb0c 2deff35d 6ca3a830
$RK_{36,37,38,39}$	4dd7cfb7 ae71c9f6 4e911fef 90aa95de
$RK_{40,41,42,43}$	2c664a7a 8cb5cf6b 14c8de1e 43b9caef
$RK_{44,45,46,47}$	568c5a33 07ef7ddd 608dc860 ac9e50f8
$RK_{48,49,50,51}$	c0c18358 4f53c80e 33e01cb9 80251e1c

plaintext		00010203	04050607	08090a0b	0c0d0e0f
initial whitening key			0f0e0d0c		0b0a0908
after whitening		00010203	0b0b0b0b	08090a0b	07070707
Round 1	input	00010203	0b0b0b0b	08090a0b	07070707
	F-function	F_0		F_1	
	input	00010203		08090a0b	
	round key	58f02029		15413cd0	
	after key add	58f1222a		1d4836db	
	after S	4ee41927		2c78a1ac	
	after M	2db2101b		d87ee718	
Round 2	input	26b91b10	08090a0b	df79e01f	00010203
	F-function	F_0		F_1	
	input	26b91b10		df79e01f	
	round key	1b0c41a4		e4bacd0f	
	after key add	3db55ab4		3bc32d10	
	after S	aa5afadb		0f1e1928	
	after M	317e029c		c0cc96ba	
Round 3	input	39770897	df79e01f	c0cd94b9	26b91b10
	F-function	F_0		F_1	
	input	39770897		c0cd94b9	
	round key	6c498393		8846231b	
	after key add	553e8b04		488bb7a2	
	after S	5487484e		d84876a0	
	after M	c3a7ac1d		7ae05884	
Round 4	input	1cde4c02	c0cd94b9	5c594394	39770897
	F-function	F_0		F_1	
	input	1cde4c02		5c594394	
	round key	1fc716fc		7c81a45b	
	after key add	03195afe		20d8e7cf	
	after S	c607fa95		12f002c9	
	after M	5edee0ce		4cfb0e90	
Round 5	input	9e137477	5c594394	758c0607	1cde4c02
	F-function	F_0		F_1	
	input	9e137477		758c0607	
	round key	fa37c259		0e3da2ee	
	after key add	6424b62e		7bb1a4e9	
	after S	4592c8d2		46f3a044	
	after M	adfd33ae		42450650	
Round 6	input	f1a4703a	758c0607	5e9b4a52	9e137477
	F-function	F_0		F_1	
	input	f1a4703a		5e9b4a52	
	round key	aacf9abb		8ec0aad9	
	after key add	5b6bea81		d05be08b	
	after S	22285e04		f822d448	
	after M	0fa52ed4		aa7a0a9c	

Round 7	input	7a2928d3 5e9b4a52	34697eeb f1a4703a
	F-function	F_0	F_1
	input	7a2928d3	34697eeb
	round key	b05bd737	8de1f2d0
	after key add	ca72ffe4	b9888c3b
	after S	23ed8e68	172b59c0
	after M	8b158630	334e2af2
Round 8	input	d58ecc62 34697eeb	c2ea5ac8 7a2928d3
	F-function	F_0	F_1
	input	d58ecc62	c2ea5ac8
	round key	8ffee0f6	b70b47ea
	after key add	5a702c94	75e11d22
	after S	facf9d64	586f2c19
	after M	72c2027e	a582d5f0
Round 9	input	46ab7c95 c2ea5ac8	dfabfd23 d58ecc62
	F-function	F_0	F_1
	input	46ab7c95	dfabfd23
	round key	581b3e34	03263f89
	after key add	1eb042a1	dc8dc2aa
	after S	177afd6a	57664735
	after M	51d5740a	110287d7
Round 10	input	933f2ec2 dfabfd23	c48c4bb5 46ab7c95
	F-function	F_0	F_1
	input	933f2ec2	c48c4bb5
	round key	2f7100cd	05cee171
	after key add	bc4e2e0f	c142aac4
	after S	e0434cd9	22fd2380
	after M	a768d32a	b6ae4f2b
Round 11	input	78c32e09 c48c4bb5	f00533be 933f2ec2
	F-function	F_0	F_1
	input	78c32e09	f00533be
	round key	b523d4e9	176d7c44
	after key add	cde0fae0	e7684ffa
	after S	3fd410d4	02ef5310
	after M	08bd9b01	2fdb3f65
Round 12	input	cc31d0b4 f00533be	bce411a7 78c32e09
	F-function	F_0	F_1
	input	cc31d0b4	bce411a7
	round key	6d7ba5d7	f797b2f3
	after key add	a14a7563	4b73a354
	after S	1b512562	c94a71eb
	after M	7c2c762b	81ca0b59

Round 13	input	8c294595 bce411a7	f9092550 cc31d0b4
	F-function	F_0	F_1
	input	8c294595	f9092550
	round key	25d80df2	a646bba2
	after key add	a9f14867	5f4f9ef2
	after S	93e47852	5c26cae5
	after M	4a87c858	54bc68d5
Round 14	input	f663d9ff f9092550	988db861 8c294595
	F-function	F_0	F_1
	input	f663d9ff	988db861
	round key	6a3a95e1	3e3a47f0
	after key add	9c594c1e	a6b7ff91
	after S	58ff39b0	054d1d75
	after M	d82301d4	085d5025
Round 15	input	212a2484 988db861	847415b0 f663d9ff
	F-function	F_0	F_1
	input	212a2484	847415b0
	round key	b304eb20	44f8824e
	after key add	922ecfa4	c08c97fe
	after S	86d2c9a0	b5ff567d
	after M	dbf56073	87e2a6a2
Round 16	input	4378d812 847415b0	71817f5d 212a2484
	F-function	F_0	F_1
	input	4378d812	71817f5d
	round key	c7557cbc	47401e21
	after key add	842da4ae	36c1617c
	after S	9e19b889	a10c5414
	after M	6791a3e3	e177d3a8
Round 17	input	e3e5b653 71817f5d	c05df72c 4378d812
	F-function	F_0	F_1
	input	e3e5b653	c05df72c
	round key	d71ff7e9	aca1fb0c
	after key add	34fa41ba	6cfc0c20
	after S	d4e1be2d	32bc13bf
	after M	2743ef2d	6fec0aab
Round 18	input	56c29070 c05df72c	2c94d2b9 e3e5b653
	F-function	F_0	F_1
	input	56c29070	2c94d2b9
	round key	2deff35d	6ca3a830
	after key add	7b2d632d	40377a89
	after S	56193719	fb13c1b7
	after M	ee6316fa	5e3245b7

Round 19	input	2e3ee1d6	2c94d2b9	bdd7f3e4	56c29070
	F-function	F_0		F_1	
	input	2e3ee1d6		bdd7f3e4	
	round key	4dd7cfb7		ae71c9f6	
	after key add	63e92e61		13a63a12	
	after S	373c4c54		8fe6c54b	
	after M	87aab08e		8f8d16f3	
Round 20	input	ab3e6237	bdd7f3e4	d94f8683	2e3ee1d6
	F-function	F_0		F_1	
	input	ab3e6237		d94f8683	
	round key	4e911fef		90aa95de	
	after key add	e5af7dd8		49e5135d	
	after S	f6ad88be		65f68f77	
	after M	0889df33		f418c84f	
Round 21	input	b55e2cd7	d94f8683	da262999	ab3e6237
	F-function	F_0		F_1	
	input	b55e2cd7		da262999	
	round key	2c664a7a		8cb5cf6b	
	after key add	993866ad		5693e6f2	
	after S	2c2b6cee		0df150e5	
	after M	8999e772		da5415d2	
Round 22	input	50d661f1	da262999	716a77e5	b55e2cd7
	F-function	F_0		F_1	
	input	50d661f1		716a77e5	
	round key	14c8de1e		43b9caef	
	after key add	441ebfef		32d3bd0a	
	after S	12b052ac		c7bbb182	
	after M	f5efd89e		744a9ced	
Round 23	input	2fc9f107	716a77e5	c114b03a	50d661f1
	F-function	F_0		F_1	
	input	2fc9f107		c114b03a	
	round key	568c5a33		07ef7ddd	
	after key add	7945ab34		c6fbcde7	
	after S	a2a77e2a		4cd7e238	
	after M	e84f6d9b		ce67e20a	
Round 24	input	99251a7e	c114b03a	9eb183fb	2fc9f107
	F-function	F_0		F_1	
	input	99251a7e		9eb183fb	
	round key	608dc860		ac9e50f8	
	after key add	f9a8d21e		322fd303	
	after S	f84572b0		c7d8f1c6	
	after M	20634b77		591b3f55	

Round 25	input	e177fb4d	9eb183fb	76d2ce52	99251a7e
	F-function	F_0		F_1	
	input	e177fb4d		76d2ce52	
	round key	c0c18358		4f53c80e	
	after key add	21b67815		3981065c	
	after S	a14dd39c		c8e20aa5	
	after M	3f88fbef		89ff5caf	
Round 26	input	a1397814	76d2ce52	10da46d1	e177fb4d
	F-function	F_0		F_1	
	input	a1397814		10da46d1	
	round key	33e01cb9		80251e1c	
	after key add	92d964ad		90ff58cd	
	after S	864445ee		9a8e803f	
	after M	5949235a		183d49c7	
output		a1397814	2f9bed08	10da46d1	f94ab28a
final whitening key			07060504		03020100
after whitening		a1397814	289de80c	10da46d1	fa48b38a
ciphertext		a1397814	289de80c	10da46d1	fa48b38a

第3章　実装手法

本章では CLEFIA のソフトウェアおよびハードウェアにおける実装手法について述べる．

3.1　ソフトウェア実装

本節では CLEFIA のソフトウェア実装手法について述べる．CLEFIA は 32 ビット及び 64 ビットプロセッサを含むさまざまプラットフォーム上のソフトウェア実装において，効率的な実装をすることが可能である．

3.1.1　暗号化の最適化手法

本節では 128 ビット鍵 CLEFIA の暗号化の実装手法について説明する．192/256 ビット鍵 CLEFIA に関しては，ラウンド数の違いを除いて同じ手法が適用できるためここでは言及しない．

32 ビットプロセッサ上の実装方法としてタイプ 1，2 の 2 種類の手法 64 ビットプロセッサ上の実装方法としてタイプ 3, 4, 5, 6 の 4 種類の手法，計 6 種類の手法について説明を行なう．まず，本節で用いる表記を以下に示す．

表記

説明を簡単にするために F 関数から鍵の加算 (鍵加算層) をはずしたものを考える．鍵加算層のない F 関数 F_0 の入力を (x_0, x_1, x_2, x_3)，出力を (y_0, y_1, y_2, y_3) とする．同様に鍵加算層のない F_1 の入力を (x_4, x_5, x_6, x_7)，出力を (y_4, y_5, y_6, y_7) とする．このとき，入出力の関係は以下のように表

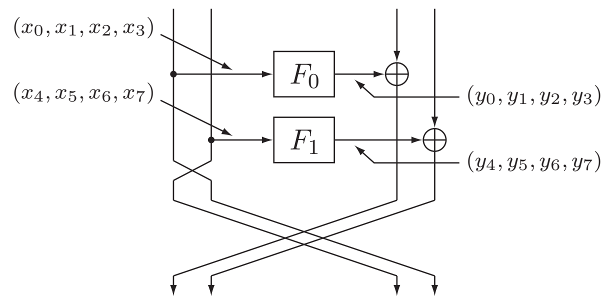

図 3.1: 64 ビットプロセッサ向け実装におけるラウンド関数の等価変形

現できる.

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} \texttt{01} & \texttt{02} & \texttt{04} & \texttt{06} \\ \texttt{02} & \texttt{01} & \texttt{06} & \texttt{04} \\ \texttt{04} & \texttt{06} & \texttt{01} & \texttt{02} \\ \texttt{06} & \texttt{04} & \texttt{02} & \texttt{01} \end{pmatrix} \begin{pmatrix} S_0(x_0) \\ S_1(x_1) \\ S_0(x_2) \\ S_1(x_3) \end{pmatrix}$$

$$\begin{pmatrix} y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} \texttt{01} & \texttt{08} & \texttt{02} & \texttt{0A} \\ \texttt{08} & \texttt{01} & \texttt{0A} & \texttt{02} \\ \texttt{02} & \texttt{0A} & \texttt{01} & \texttt{08} \\ \texttt{0A} & \texttt{02} & \texttt{08} & \texttt{01} \end{pmatrix} \begin{pmatrix} S_1(x_4) \\ S_0(x_5) \\ S_1(x_6) \\ S_0(x_7) \end{pmatrix}$$

なお，上記行列の要素はそれぞれ 16 進数表現で記述されている.

また，64 ビットプロセッサ上の実装方法においては，2 つの 32 ビットのデータ列を同時に処理することができるため，128 ビット鍵 CLEFIA のラウンド関数を図 3.1 に示すように等価変形して考える.

タイプ 1

タイプ 1 は 32 ビットプロセッサ向けの実装である．32 ビットプロセッサが十分大きな 1 次キャッシュを持っている場合，各 F 関数ごとに入力 8 ビット，出力 32 ビットの 4 つのテーブルをそれぞれ持つことで高速化できる．この実装では下記に示す 8 個のテーブルが必要である.

$$
\begin{array}{lcl}
T_{00}(x) &=& (\quad S_0(x),\ \{\mathtt{02}\}\times S_0(x),\ \{\mathtt{04}\}\times S_0(x),\ \{\mathtt{06}\}\times S_0(x)\,)\\
T_{01}(x) &=& (\,\{\mathtt{02}\}\times S_1(x),\ S_1(x),\ \{\mathtt{06}\}\times S_1(x),\ \{\mathtt{04}\}\times S_1(x)\,)\\
T_{02}(x) &=& (\,\{\mathtt{04}\}\times S_0(x),\ \{\mathtt{06}\}\times S_0(x),\ S_0(x),\ \{\mathtt{02}\}\times S_0(x)\,)\\
T_{03}(x) &=& (\,\{\mathtt{06}\}\times S_1(x),\ \{\mathtt{04}\}\times S_1(x),\ \{\mathtt{02}\}\times S_1(x),\ S_1(x)\,)\\
T_{10}(x) &=& (\quad S_1(x),\ \{\mathtt{08}\}\times S_1(x),\ \{\mathtt{02}\}\times S_1(x),\ \{\mathtt{0A}\}\times S_1(x)\,)\\
T_{11}(x) &=& (\,\{\mathtt{08}\}\times S_0(x),\ S_0(x),\ \{\mathtt{0A}\}\times S_0(x),\ \{\mathtt{02}\}\times S_0(x)\,)\\
T_{12}(x) &=& (\,\{\mathtt{02}\}\times S_1(x),\ \{\mathtt{0A}\}\times S_1(x),\ S_1(x),\ \{\mathtt{08}\}\times S_1(x)\,)\\
T_{13}(x) &=& (\,\{\mathtt{0A}\}\times S_0(x),\ \{\mathtt{02}\}\times S_0(x),\ \{\mathtt{08}\}\times S_0(x),\ S_0(x)\,)
\end{array}
$$

次に以下の式を計算する．

$$
\begin{array}{rcl}
(y_0, y_1, y_2, y_3) &=& T_{00}(x_0) \oplus T_{01}(x_1) \oplus T_{02}(x_2) \oplus T_{03}(x_3)\\
(y_4, y_5, y_6, y_7) &=& T_{10}(x_4) \oplus T_{11}(x_5) \oplus T_{12}(x_6) \oplus T_{13}(x_7)
\end{array}
$$

この実装に必要な演算の見積もりを下記に示す．

テーブルサイズ (KB):	8
ラウンドあたりの演算回数 (全ラウンド数 18)	
テーブル参照:	8
XOR (F 関数内):	6
XOR (F 関数外):	2
XOR (鍵加算):	2
XOR (鍵ホワイトニング):	4

タイプ 2

32 ビットプロセッサ上でのタイプ 1 とは異なる手法として，巡回シフト演算を利用しタイプ 1 のテーブルサイズを半分に削減する方法を説明する．この実装はプロセッサの 1 次キャッシュが 8KB より小さく，かつ巡回シフト演算が比較的高速である場合に有効である．

この実装ではタイプ 1 で必要であったテーブル $T_{02}(x), T_{03}(x), T_{12}(x), T_{13}(x)$ を事前に持つ必要がなくなる．これらのテーブルは他のテーブルにより，以下のように計算することができる．

$$
\begin{array}{rcl}
T_{02}(x) &=& T_{00}(x) \lll 16\\
T_{03}(x) &=& T_{01}(x) \lll 16\\
T_{12}(x) &=& T_{10}(x) \lll 16\\
T_{13}(x) &=& T_{11}(x) \lll 16
\end{array}
$$

この実装に必要な演算の見積もりを下記に示す．

テーブルサイズ (KB):	4
ラウンドあたりの演算回数 (全ラウンド数 18)	
テーブル参照:	8
巡回シフト演算:	4
XOR (F 関数内):	6
XOR (F 関数外):	2
XOR (鍵加算):	2
XOR (鍵ホワイトニング):	4

タイプ 1 の実装見積りと比較してテーブルサイズが 8KB から 4KB に削減されている．

タイプ 3

タイプ 3 は 64 ビットプロセッサ向けの実装である．64 ビットプロセッサが 16KB 以上の 1 次キャッシュを持つ場合，ラウンド関数に以下に示す 8 ビット入力 64 ビット出力の 8 つのテーブルを使うことができる．

$$\begin{aligned}
T_{00}(x) &= (\ S_0(x),\ \{\texttt{02}\}\times S_0(x),\ \{\texttt{04}\}\times S_0(x),\ \{\texttt{06}\}\times S_0(x),\ 0,\ 0,\ 0,\ 0\,)\\
T_{01}(x) &= (\,\{\texttt{02}\}\times S_1(x),\ S_1(x),\ \{\texttt{06}\}\times S_1(x),\ \{\texttt{04}\}\times S_1(x),\ 0,\ 0,\ 0,\ 0\,)\\
T_{02}(x) &= (\,\{\texttt{04}\}\times S_0(x),\ \{\texttt{06}\}\times S_0(x),\ S_0(x),\ \{\texttt{02}\}\times S_0(x),\ 0,\ 0,\ 0,\ 0\,)\\
T_{03}(x) &= (\,\{\texttt{06}\}\times S_1(x),\ \{\texttt{04}\}\times S_1(x),\ \{\texttt{02}\}\times S_1(x),\ S_1(x),\ 0,\ 0,\ 0,\ 0\,)
\end{aligned}$$

$$\begin{aligned}
T_{10}(x) &= (\,0,\ 0,\ 0,\ 0,\ S_1(x),\ \{\texttt{08}\}\times S_1(x),\ \{\texttt{02}\}\times S_1(x),\ \{\texttt{0A}\}\times S_1(x)\,)\\
T_{11}(x) &= (\,0,\ 0,\ 0,\ 0,\ \{\texttt{08}\}\times S_0(x),\ S_0(x),\ \{\texttt{0A}\}\times S_0(x),\ \{\texttt{02}\}\times S_0(x)\,)\\
T_{12}(x) &= (\,0,\ 0,\ 0,\ 0,\ \{\texttt{02}\}\times S_1(x),\ \{\texttt{0A}\}\times S_1(x),\ S_1(x),\ \{\texttt{08}\}\times S_1(x)\,)\\
T_{13}(x) &= (\,0,\ 0,\ 0,\ 0,\ \{\texttt{0A}\}\times S_0(x),\ \{\texttt{02}\}\times S_0(x),\ \{\texttt{08}\}\times S_0(x),\ S_0(x)\,)
\end{aligned}$$

これらのテーブルを使い，以下の式を計算する．

$$\begin{aligned}
(y_0, y_1, y_2, y_3, y_4, y_5, y_6, y_7) &= T_{00}(x_0) \oplus T_{01}(x_1) \oplus T_{02}(x_2) \oplus T_{03}(x_3) \oplus\\
&\quad\ T_{10}(x_4) \oplus T_{11}(x_5) \oplus T_{12}(x_6) \oplus T_{13}(x_7)
\end{aligned}$$

この実装に必要な演算を以下に示す．

テーブルサイズ (KB):	16
ラウンドあたりの演算回数 (全ラウンド数 18)	
テーブル参照:	8
XOR (F 関数内):	7
XOR (F 関数外):	1
XOR (鍵加算):	1
スワップ演算 (巡回シフト演算):	1
XOR(鍵ホワイトニング):	2

図 3.1 からわかるように (x_0, x_1, x_2, x_3) と (x_4, x_5, x_6, x_7) をスワップするために巡回シフト演算が必要である．

タイプ 4

64 ビットプロセッサが例えば 32KB のような十分大きい 1 次キャッシュをもっている場合，下記のテーブルを追加することにより，タイプ 3 で必要であったスワップ演算を回避することができる．

$$
\begin{array}{lcl}
T_{04}(x) & = & (0,\ 0,\ 0,\ 0,\ S_0(x),\ \{\texttt{02}\}\times S_0(x),\ \{\texttt{04}\}\times S_0(x),\ \{\texttt{06}\}\times S_0(x))\\
T_{05}(x) & = & (0,\ 0,\ 0,\ 0,\ \{\texttt{02}\}\times S_1(x),\ S_1(x),\ \{\texttt{06}\}\times S_1(x),\ \{\texttt{04}\}\times S_1(x))\\
T_{06}(x) & = & (0,\ 0,\ 0,\ 0,\ \{\texttt{04}\}\times S_0(x),\ \{\texttt{06}\}\times S_0(x),\ S_0(x),\ \{\texttt{02}\}\times S_0(x))\\
T_{07}(x) & = & (0,\ 0,\ 0,\ 0,\ \{\texttt{06}\}\times S_1(x),\ \{\texttt{04}\}\times S_1(x),\ \{\texttt{02}\}\times S_1(x),\ S_1(x))
\end{array}
$$

$$
\begin{array}{lcl}
T_{14}(x) & = & (S_1(x),\ \{\texttt{08}\}\times S_1(x),\ \{\texttt{02}\}\times S_1(x),\ \{\texttt{0A}\}\times S_1(x),\ 0,\ 0,\ 0,\ 0)\\
T_{15}(x) & = & (\{\texttt{08}\}\times S_0(x),\ S_0(x),\ \{\texttt{0A}\}\times S_0(x),\ \{\texttt{02}\}\times S_0(x),\ 0,\ 0,\ 0,\ 0)\\
T_{16}(x) & = & (\{\texttt{02}\}\times S_1(x),\ \{\texttt{0A}\}\times S_1(x),\ S_1(x),\ \{\texttt{08}\}\times S_1(x),\ 0,\ 0,\ 0,\ 0)\\
T_{17}(x) & = & (\{\texttt{0A}\}\times S_0(x),\ \{\texttt{02}\}\times S_0(x),\ \{\texttt{08}\}\times S_0(x),\ S_0(x),\ 0,\ 0,\ 0,\ 0)
\end{array}
$$

ラウンド毎に下記の式の一方を適切に選択する．

$$
\begin{aligned}
(y_0,y_1,y_2,y_3,y_4,y_5,y_6,y_7) &= T_{00}(x_0)\oplus T_{01}(x_1)\oplus T_{02}(x_2)\oplus T_{03}(x_3)\oplus\\
&\quad T_{10}(x_4)\oplus T_{11}(x_5)\oplus T_{12}(x_6)\oplus T_{13}(x_7)\\
(y_0,y_1,y_2,y_3,y_4,y_5,y_6,y_7) &= T_{04}(x_0)\oplus T_{05}(x_1)\oplus T_{06}(x_2)\oplus T_{07}(x_3)\oplus\\
&\quad T_{14}(x_4)\oplus T_{15}(x_5)\oplus T_{16}(x_6)\oplus T_{17}(x_7)
\end{aligned}
$$

この実装では下記の演算が必要である．

テーブルサイズ (KB):	32
ラウンドあたりの演算回数 (全ラウンド数 18)	
テーブル参照:	8
XOR (F 関数内):	7
XOR (F 関数外):	1
XOR (鍵加算):	1
XOR (鍵ホワイトニング):	2

この実装のタイプ 3 に対する利点は，スワップ演算が必要のない点である．

タイプ 5

タイプ 5 で用いたテーブル T_{0i}, T_{1i} $(0\le i\le 3)$ を下記に示す．テーブル T_{2i} $(0\le i\le 3)$ にマージすることにより，テーブルサイズを半分に削

減する実装法について説明する．

$$
\begin{array}{rcl}
T_{20}(x) & = & (\quad S_0(x),\ S_1(x),\ \{\mathtt{02}\}\times S_0(x),\ \{\mathtt{08}\}\times S_1(x), \\
& & \{\mathtt{04}\}\times S_0(x),\ \{\mathtt{02}\}\times S_1(x),\ \{\mathtt{06}\}\times S_0(x),\ \{\mathtt{0A}\}\times S_1(x)) \\
T_{21}(x) & = & (\{\mathtt{02}\}\times S_1(x),\ \{\mathtt{08}\}\times S_0(x),\ S_1(x),\ S_0(x), \\
& & \{\mathtt{06}\}\times S_1(x),\ \{\mathtt{0A}\}\times S_0(x),\ \{\mathtt{04}\}\times S_1(x),\ \{\mathtt{02}\}\times S_0(x)) \\
T_{22}(x) & = & (\{\mathtt{04}\}\times S_0(x),\ \{\mathtt{02}\}\times S_1(x),\ \{\mathtt{06}\}\times S_0(x),\ \{\mathtt{0A}\}\times S_1(x), \\
& & S_0(x),\ S_1(x),\ \{\mathtt{02}\}\times S_0(x),\ \{\mathtt{08}\}\times S_1(x)) \\
T_{23}(x) & = & (\{\mathtt{06}\}\times S_1(x),\ \{\mathtt{0A}\}\times S_0(x),\ \{\mathtt{04}\}\times S_1(x),\ \{\mathtt{02}\}\times S_0(x), \\
& & \{\mathtt{02}\}\times S_1(x),\ \{\mathtt{08}\}\times S_0(x),\ S_1(x),\ S_0(x))
\end{array}
$$

以下のマスク処理を含んだ下記の式を計算する．

$$
\begin{array}{rl}
& (y_0, y_1, y_2, y_3, y_4, y_5, y_6, y_7) \\
= & (T_{20}(x_0) \oplus T_{21}(x_1) \oplus T_{22}(x_2) \oplus T_{23}(x_3))\ \&\ \mathtt{0xFF00FF00FF00FF00}) \oplus \\
& (T_{20}(x_4) \oplus T_{21}(x_5) \oplus T_{22}(x_6) \oplus T_{23}(x_7))\ \&\ \mathtt{0x00FF00FF00FF00FF})
\end{array}
$$

この実装で必要な演算は下記である．

テーブルサイズ (KB):	8
ラウンドあたりの演算回数 (全ラウンド数 18)	
テーブル参照:	8
XOR (F 関数内):	7
XOR (F 関数外):	1
AND (マスク演算):	2
XOR (鍵加算):	1
スワップ演算 (巡回シフト演算):	1
XOR (鍵ホワイトニング):	2

タイプ 6

タイプ 5 のテーブルサイズは，巡回シフト演算を用いることにより削減することができる．この実装方法はプロセッサの 1 次キャッシュが小さい時に有効である．タイプ 5 で必要であった $T_{22}(x)$ 及び $T_{23}(x)$ のテーブルは必要ではなく，F 関数の出力は以下のように他のテーブルを用いて生成することができる．

$$
\begin{array}{rcl}
T_{22}(x) & = & T_{20}(x) \lll 32 \\
T_{23}(x) & = & T_{21}(x) \lll 32
\end{array}
$$

この実装には下記の演算が必要である．

テーブルサイズ (KB):	4
ラウンドあたりの演算回数 (全ラウンド数 18)	
テーブル参照:	8
XOR (F 関数内):	7
XOR (F 関数外):	1
AND (マスク演算):	2
巡回シフト演算:	4
XOR (鍵加算):	1
スワップ演算 (巡回シフト演算):	1
XOR (鍵加算):	2

3.1.2 復号の最適化手法

2.2 節及び 2.3.1 節で述べたように CLEFIA は完全な involution 性を有していない．暗号化関数と復号関数を別々に実装した場合，速度に関して最大の性能が得られる．一方，コードサイズやその他の理由により暗復号関数を 1 つにマージする必要がある場合，下記の手法を用いることで実装速度の大きな低下を防ぐことができる．

- 偶数ラウンドにおいて，F 関数の参照テーブルのアドレスを暗号化もしくは復号で変更する．
- 偶数ラウンドで入力するラウンド鍵を F 関数にあわせて変更する．
- 復号の場合，最終出力をスワップする．

3.1.3 鍵スケジューリングの最適化手法

鍵スケジューリング演算は 2 つの要素から構成されている．秘密鍵 K から L を生成する部分，K と L から拡大鍵を生成する部分である．前者は暗号化のラウンド関数を利用する．従って，前節で述べたラウンド関数と同じ最適化手法が適用可能である．後者は，巡回シフト演算を含む比較的軽い演算で実現できる．

3.2 ハードウェア実装

本節では，CLEFIA のハードウェア実装手法として，F 関数と鍵スケジュール部の最適化手法について述べる．

3.2.1 F 関数の最適化手法

本節では，F 関数の最適化手法として，S-box S_0, S_1 および拡散行列 M_0, M_1 の実装法について検討する．

S-box S_0

8 ビット入出力の S-box S_0 は，4 ビット入出力 S-box SS_0, SS_1 と SS_2, SS_3 の間に，原始多項式 z^4+z+1 で定義される GF(2^4) 上の演算で構成された線形変換層を挟む形で構成される．4 ビット入出力 S-box は 16 入力 $\times$ 4 ビットのテーブルから論理合成ツールを用いて自動合成することができる．また，GF(2^4) 上の演算に関しては，`0x2` 倍演算が 1 個の XOR (排他的論理和) ゲート，加算が 4 個の XOR ゲートで実装可能である．従って，線形変換層は計 10 個の XOR ゲートで構成可能であり，最大遅延は 2 XOR gate となる．

S-box S_1

8 ビット入出力の S-box S_1 は，アフィン変換 f，原始多項式 $p(z) = z^8+z^4+z^3+z^2+1$ で定義される GF(2^8) 上の逆元演算，アフィン変換 g を順に適用することで構成される．同様の構成を持つ AES の S-box の実装では，GF(2^8) 上の逆元演算の代わりに合成体 GF($(2^4)^2$) 上の逆元演算を用いることにより，ゲート規模を大幅に削減できることが知られている [6]. そこで，我々は次のような式で定義される合成体 GF($(2^4)^2$) を選択することにより S_1 の小型化を図った．

$$\begin{cases} \mathrm{GF}(2^4) & : q_0(z) = z^4+z+1 \\ \mathrm{GF}((2^4)^2) & : q_1(z) = z^2+z+\lambda \quad (\lambda = \omega^3) \end{cases}$$

ここで，ω は $q_0(z)=0$ の根である．また，$\mathrm{GF}(2^8)$ から $\mathrm{GF}((2^4)^2)$ への同型写像として次のような写像 ϕ を選択した．

$$\begin{aligned}\phi &: x_0\alpha^7+x_1\alpha^6+x_2\alpha^5+x_3\alpha^4+x_4\alpha^3+x_5\alpha^2+x_6\alpha+x_7\\ &\mapsto (y_0\omega^3+y_1\omega^2+y_2\omega+y_3)\beta+(y_4\omega^3+y_5\omega^2+y_6\omega+y_7)\end{aligned}$$

$$\begin{pmatrix} y_0\\ y_1\\ y_2\\ y_3\\ y_4\\ y_5\\ y_6\\ y_7 \end{pmatrix} = \begin{pmatrix} 0&0&1&0&0&0&0&0\\ 0&0&1&1&1&0&1&0\\ 0&1&1&0&0&1&0&0\\ 1&1&1&1&0&1&0&0\\ 1&0&1&0&1&1&0&0\\ 1&0&1&0&1&1&1&0\\ 0&0&0&0&1&0&1&0\\ 1&1&1&0&1&1&1&1 \end{pmatrix} \begin{pmatrix} x_0\\ x_1\\ x_2\\ x_3\\ x_4\\ x_5\\ x_6\\ x_7 \end{pmatrix}$$

ここで，α を $q_0(z)=0$，β を $q_1(z)=0$ の根とする．また，$\mathrm{GF}((2^4)^2)$ から $\mathrm{GF}(2^8)$ への逆同型写像 ϕ^{-1} は以下のように表せる．

$$\begin{aligned}\phi^{-1} &: (x_0\omega^3+x_1\omega^2+x_2\omega+x_3)\beta+(x_4\omega^3+x_5\omega^2+x_6\omega+x_7)\\ &\mapsto y_0\alpha^7+y_1\alpha^6+y_2\alpha^5+y_3\alpha^4+y_4\alpha^3+y_5\alpha^2+y_6\alpha+y_7\end{aligned}$$

$$\begin{pmatrix} y_0\\ y_1\\ y_2\\ y_3\\ y_4\\ y_5\\ y_6\\ y_7 \end{pmatrix} = \begin{pmatrix} 1&1&1&1&0&0&1&0\\ 1&1&0&1&0&1&0&0\\ 1&0&0&0&0&0&0&0\\ 1&1&0&0&0&0&1&0\\ 0&0&0&0&1&1&1&0\\ 0&1&1&1&0&1&0&0\\ 0&0&0&0&1&1&0&0\\ 1&1&0&1&0&0&0&1 \end{pmatrix} \begin{pmatrix} x_0\\ x_1\\ x_2\\ x_3\\ x_4\\ x_5\\ x_6\\ x_7 \end{pmatrix}$$

アフィン変換 f と同型写像 ϕ は次の行列演算に合成することができる．

$$\begin{aligned}\phi\circ f &: x_0\alpha^7+x_1\alpha^6+x_2\alpha^5+x_3\alpha^4+x_4\alpha^3+x_5\alpha^2+x_6\alpha+x_7\\ &\mapsto (y_0\omega^3+y_1\omega^2+y_2\omega+y_3)\beta+(y_4\omega^3+y_5\omega^2+y_6\omega+y_7)\end{aligned}$$

$$\begin{pmatrix} y_0\\ y_1\\ y_2\\ y_3\\ y_4\\ y_5\\ y_6\\ y_7 \end{pmatrix} = \begin{pmatrix} 0&0&0&0&0&0&0&1\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&1&1&0&0\\ 0&0&0&1&0&0&1&0\\ 0&0&1&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&0&0&0&1&0&1\\ 1&0&0&1&0&0&0&0 \end{pmatrix} \begin{pmatrix} x_0\\ x_1\\ x_2\\ x_3\\ x_4\\ x_5\\ x_6\\ x_7 \end{pmatrix} + \begin{pmatrix} 0\\ 1\\ 1\\ 0\\ 0\\ 1\\ 0\\ 1 \end{pmatrix}$$

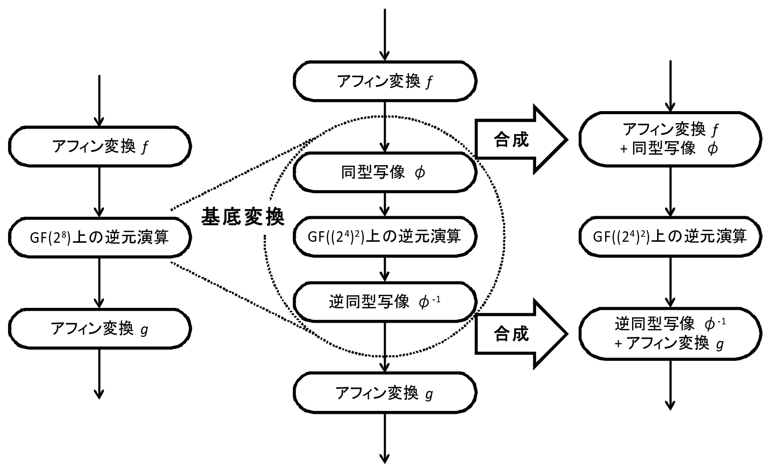

図 3.2: S-box S_1 の最適化実装

同様に，逆同型写像 ϕ^{-1} とアフィン変換 g も次の行列演算に合成することができる．

$$
\begin{aligned}
g \circ \phi^{-1} : (x_0\omega^3 + x_1\omega^2 + x_2\omega + x_3)\beta + (x_4\omega^3 + x_5\omega^2 + x_6\omega + x_7) \\
\mapsto y_0\alpha^7 + y_1\alpha^6 + y_2\alpha^5 + y_3\alpha^4 + y_4\alpha^3 + y_5\alpha^2 + y_6\alpha + y_7
\end{aligned}
$$

$$
\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix}
=
\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}
+
\begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}
$$

これらの行列演算 $\phi \circ f$ と $g \circ \phi^{-1}$ はそれぞれ 2 XOR + 2 XNOR + 2 NOT および 2 XOR + 4 XNOR で実装可能である．ここで，XNOR は排他的論理和の否定回路，NOT は否定回路を示している．

GF$((2^4)^2)$ 上の任意の元 $a_0\beta + a_1$ $(a_0, a_1 \in \mathrm{GF}(2^4))$ に対して，その逆元 $b_0\beta + b_1 = (a_0\beta + a_1)^{-1}$ $(b_0, b_1 \in \mathrm{GF}(2^4))$ は

$$
\begin{aligned}
b_0 &= a_0\Delta^{-1} \\
b_1 &= (a_0 + a_1)\Delta^{-1} \\
\Delta &= (a_0 + a_1)a_1 + \lambda a_0^2
\end{aligned}
$$

と計算することができ，これらの演算は全て GF(2^4) 上の演算となる．

図 3.2 に S-box S_1 の最適化手法をまとめている．

拡散行列 M_0, M_1

拡散行列 M_0, M_1 は，既約多項式 $z^8+z^4+z^3+z^2+1$ で定義される $\mathrm{GF}(2^8)$ 上の 4×4 Hadamard 行列である．$\mathrm{GF}(2^8)$ の元の加算 $\oplus$ は，元のビット同士の XOR 演算と等価であり，8 個の XOR ゲートが必要となる．また，$\mathrm{GF}(2^8)$ 上の乗算 $\times$ は $p(z)$ を法とする多項式の乗算に相当し，$\mathrm{GF}(2^8)$ の任意の元 a に対する定数乗算 $\{\texttt{02}\}\times a$, $\{\texttt{04}\}\times a$, $\{\texttt{08}\}\times a$ はそれぞれ 3, 5, 8 個の XOR ゲートを必要とする．

行列 M_0 の乗算に対する入力ベクトルを (X_0, X_1, X_2, X_3)，出力ベクトルを (Y_0, Y_1, Y_2, Y_3) とすると，行列 M_0 の乗算は，次式のように，非ゼロの要素が {01, 02, 04} のどれか 1 つのみで構成される 3 つの行列の乗算に分割することができる．

$$
\begin{aligned}
\begin{pmatrix} Y_0\\ Y_1\\ Y_2\\ Y_3 \end{pmatrix}
&= \begin{pmatrix} \texttt{01} & \texttt{02} & \texttt{04} & \texttt{06}\\ \texttt{02} & \texttt{01} & \texttt{06} & \texttt{04}\\ \texttt{04} & \texttt{06} & \texttt{01} & \texttt{02}\\ \texttt{06} & \texttt{04} & \texttt{02} & \texttt{01} \end{pmatrix}
\begin{pmatrix} X_0\\ X_1\\ X_2\\ X_3 \end{pmatrix}\\
&= \begin{pmatrix} \texttt{01} & \texttt{00} & \texttt{00} & \texttt{00}\\ \texttt{00} & \texttt{01} & \texttt{00} & \texttt{00}\\ \texttt{00} & \texttt{00} & \texttt{01} & \texttt{00}\\ \texttt{00} & \texttt{00} & \texttt{00} & \texttt{01} \end{pmatrix}
\begin{pmatrix} X_0\\ X_1\\ X_2\\ X_3 \end{pmatrix}
+ \begin{pmatrix} \texttt{00} & \texttt{02} & \texttt{00} & \texttt{02}\\ \texttt{02} & \texttt{00} & \texttt{02} & \texttt{00}\\ \texttt{00} & \texttt{02} & \texttt{00} & \texttt{02}\\ \texttt{02} & \texttt{00} & \texttt{02} & \texttt{00} \end{pmatrix}
\begin{pmatrix} X_0\\ X_1\\ X_2\\ X_3 \end{pmatrix}\\
&+ \begin{pmatrix} \texttt{00} & \texttt{00} & \texttt{04} & \texttt{04}\\ \texttt{00} & \texttt{00} & \texttt{04} & \texttt{04}\\ \texttt{04} & \texttt{04} & \texttt{00} & \texttt{00}\\ \texttt{04} & \texttt{04} & \texttt{00} & \texttt{00} \end{pmatrix}
\begin{pmatrix} X_0\\ X_1\\ X_2\\ X_3 \end{pmatrix}
\end{aligned}
$$

Hadamard 行列の特性により，次式に従って中間値を共有し，XOR ゲート数を減らすことができる．

$$
\begin{cases} A_0 = X_0 \oplus X_1\\ A_1 = X_2 \oplus X_3 \end{cases}
\begin{cases} C_0 = \{\texttt{02}\} \times B_0\\ C_1 = \{\texttt{02}\} \times B_1 \end{cases}
\begin{cases} Y_0 = C_1 \oplus D_1 \oplus X_0\\ Y_1 = C_0 \oplus D_1 \oplus X_1\\ Y_2 = C_1 \oplus D_0 \oplus X_2\\ Y_3 = C_0 \oplus D_0 \oplus X_3 \end{cases}
$$
$$
\begin{cases} B_0 = X_0 \oplus X_2\\ B_1 = X_1 \oplus X_3 \end{cases}
\begin{cases} D_0 = \{\texttt{04}\} \times A_0\\ D_1 = \{\texttt{04}\} \times A_1 \end{cases}
$$

以上より，行列 M_0 の乗算に必要な XOR ゲート数は 112 となり，遅延段数は 4 段になる．

行列 M_1 の乗算に対する入力ベクトルを (X_0, X_1, X_2, X_3)，出力ベクトルを (Y_0, Y_1, Y_2, Y_3) とすると，行列 M_1 の乗算は，次式のように，非ゼ

ロの要素が $\{\texttt{01},\ \texttt{02},\ \texttt{08}\}$ のどれか 1 つのみで構成される 3 つの行列の乗算に分割することができる．

$$
\begin{aligned}
\begin{pmatrix} Z_0 \\ Z_1 \\ Z_2 \\ Z_3 \end{pmatrix}
&= \begin{pmatrix} \texttt{01} & \texttt{08} & \texttt{02} & \texttt{0A} \\ \texttt{08} & \texttt{01} & \texttt{0A} & \texttt{02} \\ \texttt{02} & \texttt{0A} & \texttt{01} & \texttt{08} \\ \texttt{0A} & \texttt{02} & \texttt{08} & \texttt{01} \end{pmatrix}
\begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{pmatrix} \\
&= \begin{pmatrix} \texttt{01} & \texttt{00} & \texttt{00} & \texttt{00} \\ \texttt{00} & \texttt{01} & \texttt{00} & \texttt{00} \\ \texttt{00} & \texttt{00} & \texttt{01} & \texttt{00} \\ \texttt{00} & \texttt{00} & \texttt{00} & \texttt{01} \end{pmatrix}
\begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{pmatrix}
+ \begin{pmatrix} \texttt{00} & \texttt{00} & \texttt{02} & \texttt{02} \\ \texttt{00} & \texttt{00} & \texttt{02} & \texttt{02} \\ \texttt{02} & \texttt{02} & \texttt{00} & \texttt{00} \\ \texttt{02} & \texttt{02} & \texttt{00} & \texttt{00} \end{pmatrix}
\begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{pmatrix} \\
&+ \begin{pmatrix} \texttt{00} & \texttt{08} & \texttt{00} & \texttt{08} \\ \texttt{08} & \texttt{00} & \texttt{08} & \texttt{00} \\ \texttt{00} & \texttt{08} & \texttt{00} & \texttt{08} \\ \texttt{08} & \texttt{00} & \texttt{08} & \texttt{00} \end{pmatrix}
\begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{pmatrix}
\end{aligned}
$$

M_0 の乗算と同様に，次式に従って中間値を共有することができる．

$$
\begin{cases} A_0 = X_0 \oplus X_1 \\ A_1 = X_2 \oplus X_3 \end{cases}
\begin{cases} C_0 = \{\texttt{02}\} \times A_0 \\ C_1 = \{\texttt{02}\} \times A_1 \end{cases}
\begin{cases} Z_0 = C_1 \oplus D_1 \oplus X_0 \\ Z_1 = C_1 \oplus D_0 \oplus X_1 \\ Z_2 = C_0 \oplus D_1 \oplus X_2 \\ Z_3 = C_0 \oplus D_0 \oplus X_3 \end{cases}
$$
$$
\begin{cases} B_0 = X_0 \oplus X_2 \\ B_1 = X_1 \oplus X_3 \end{cases}
\begin{cases} D_0 = \{\texttt{08}\} \times B_0 \\ D_1 = \{\texttt{08}\} \times B_1 \end{cases}
$$

以上より，行列 M_1 の乗算に必要な XOR ゲート数は 118 となり，遅延段数は行列 M_0 と同様に 4 段となる．

なお，2 つの F 関数 F_0 と F_1 を共有する小型化実装においては行列 M_0, M_1 を共有化することになるが，共有化回路に必要な XOR ゲート数と 2:1 セレクタゲート数はそれぞれ 188, 32 となり，最大遅延は XOR ゲート 4 段と 2:1 セレクタゲート 1 段分となる．

3.2.2　鍵スケジューリング部の最適化手法

本節では，CLEFIA の鍵スケジュール部のハードウェア実装における最適化手法について簡単に紹介する．詳細については，文献 [9] に述べられている．CLEFIA の鍵スケジュール部は大きく分けて，秘密鍵 K より中間鍵 L を生成 (ステップ 1)，K と L より WK_i および RK_j を生成 (ステップ 2)，という 2 つの処理により構成される．

ステップ 1 は，データ処理部を用いることで回路をほとんど追加することなく実行することが可能である．128 ビット鍵の場合には，秘密鍵 K を入力，定数値をラウンド鍵とした $GFN_{4,12}$ は暗号化とラウンド関

数を共有することにより実装可能である．192/256 ビット鍵の場合には，$GFN_{8,10}$ は $GFN_{4,r}$ と F 関数 F_0, F_1 を共有することにより実装可能となる．ステップ 2 では，中間鍵レジスタが 2 ラウンドに 1 回 $DoubleSwap$ 関数により更新される．32 ビットのラウンド鍵 RK_j は，中間鍵レジスタのうちの 32 ビットにラウンド定数と，適応的に選択される秘密鍵 K の一部を XOR することにより生成される．従って，暗号化/復号時にも秘密鍵 K を鍵入力として固定した状態であれば，128 ビット鍵の場合には 128 ビット，192/256 ビット鍵の場合には 256 ビットの中間鍵レジスタのみで，鍵スケジュール部を実装することが可能である．

以降では，128 ビット鍵の CLEFIA に対する鍵スケジュール部の更なる最適化手法として $DoubleSwap$ 関数の実装法について考察を行なう．なお，192/256 ビット鍵 CLEFIA の鍵スケジュール部にも同様の最適化手法を適用することができる．$DoubleSwap$ 関数 Σ は 128 ビット入出力の置換関数であり，

$$
\begin{aligned}
&\Sigma : X \mapsto Y \\
&\quad Y = X[7\text{-}63] \mid X[121\text{-}127] \mid X[0\text{-}6] \mid X[64\text{-}120]
\end{aligned}
$$

で定義される．ただし，$X[a\text{-}b]$ は X の a ビット目から b ビット目を切り出したデータを表し，0 ビット目を MSB とする．

中間鍵 L は鍵セットアップ時に生成され，128 ビットの中間鍵レジスタに格納される．暗号化の一般的な実装法では，2 ラウンドに 1 回 $DoubleSwap$ 関数 Σ を適用することにより中間鍵レジスタが更新され，奇数ラウンドでは中間鍵レジスタの上位 64 ビットを，偶数ラウンドでは中間鍵レジスタの下位 64 ビットを元にラウンド鍵は生成される．暗号化の最終ラウンド後に Σ の逆関数 Σ^{-1} の 8 回繰り返しに相当する Σ^{-8} を適用することにより中間鍵レジスタを L に戻すことができる．復号の場合には，開始時に Σ の 8 回繰り返しに相当する Σ^8 を適用することにより，$\Sigma^8(L)$ が中間鍵レジスタに格納される．その後，2 ラウンドに 1 回 Σ^{-1} を適用することにより中間鍵レジスタが更新され，奇数ラウンドでは中間鍵レジスタの下位 64 ビットを，偶数ラウンドでは中間鍵レジスタの上位 64 ビットを元にラウンド鍵は生成される．以上より，暗復号において $\Sigma, \Sigma^{-1}, \Sigma^8, \Sigma^{-8}$ の 4 つの関数が必要となる．

必要な関数を削減するために，$DoubleSwap$ 関数 Σ を，次式で定義される $Swap$ 関数 Ω と $SubSwap$ 関数 Ψ を用いて $\Sigma = \Psi \circ \Omega$ のように分

解する．

$$\begin{aligned}
\Omega &: X \mapsto Y \\
&\quad Y = X[64\text{-}127] \mid X[0\text{-}63] \\
\Psi &: X \mapsto Y \\
&\quad Y = X[71\text{-}127] \mid X[57\text{-}70] \mid X[0\text{-}56]
\end{aligned}$$

Ω は上位 64 ビットと下位 64 ビットを入れ換える関数であり，また Ω, Ψ ともに自身が逆関数と一致している．暗号化時には，中間鍵レジスタが奇数ラウンドで Ω を，偶数ラウンドで Ψ を適用することにより更新されると，どのラウンドにおいてもラウンド鍵が中間鍵レジスタの上位 64 ビットを元に生成されることになる．最終ラウンド後に，次式で表わされる $FinalSwap$ 関数 Φ を適用することにより中間鍵レジスタを L に戻すことができる．

$$\begin{aligned}
\Phi &: X \mapsto Y \\
&\quad Y = X[49\text{-}55] \mid X[42\text{-}48] \mid X[35\text{-}41] \mid X[28\text{-}34] \mid X[21\text{-}27] \mid X[14\text{-}20] \mid \\
&\qquad X[7\text{-}13] \mid X[0\text{-}6] \mid X[64\text{-}71] \mid X[56\text{-}63] \mid X[121\text{-}127] \mid X[114\text{-}120] \mid \\
&\qquad X[107\text{-}113] \mid X[100\text{-}106] \mid X[93\text{-}99] \mid X[86\text{-}92] \mid X[79\text{-}85] \mid X[72\text{-}78]
\end{aligned}$$

この $FinalSwap$ 関数 Φ も自身が逆関数と一致している．復号の場合には，暗号化時の逆関数を適用すればよいが，関数 Ω, Ψ, Φ は全て自身が逆関数と一致していることから，復号開始時に Φ を，奇数ラウンド目で Ω を，偶数ラウンド目で Ψ を適用することにより，常に中間鍵レジスタの上位 64 ビットを元にラウンド鍵を生成することができる．以上より，暗復号に必要な関数が Ω, Ψ, Φ の 3 種類となり，128 ビットのセレクタを削減することができ，更に常に中間鍵レジスタの上位 64 ビットを基にラウンド鍵を生成することから 64 ビットのセレクタも削減可能となる．

第 4 章　バージョン情報

CLEFIA のアルゴリズム仕様はこの暗号技術仕様書の記載で一意に定められ，他のバージョンは存在しない．

CLEFIA は，同一の名称，かつ同一の仕様で以下に発表および公開を行っている．

論文発表および公開情報

- 電子情報通信学会 情報セキュリティ研究会
  白井, 渋谷, 秋下, 盛合, 岩田, 「128 ビットブロック暗号 CLEFIA」
  信学技報 Vol.107, No.44, pp.1-9, 2007 年 5 月 11 日.

- 国際会議 Fast Software Encryption 2007
  Taizo Shirai, Kyoji Shibutani, Toru Akishita, Shiho Moriai, Tetsu Iwata, "The 128-bit Blockcipher CLEFIA." FSE 2007, LNCS 4593, pp. 181-195, Springer-Verlag, 2007.

- IETF Internet Draft:
  M. Katagi, S. Moriai, "The 128-bit Blockcipher CLEFIA", October 19, 2009. http://tools.ietf.org/html/draft-katagi-clefia-00

- CLEFIA ウェブサイト
  Sony Corporation, "The 128-bit Blockcipher CLEFIA : Algorithm Specification, Version 1.0, 2007. http://www.sony.co.jp/clefia

第 5 章　利用実績および推奨用途

5.1　利用実績

5.1.1　標準化

以下の標準化機関に CLEFIA のアルゴリズムを提案中である．

ISO/IEC JTC 1/SC27　ISO/IEC 29192 – Information technology – Security techniques – Lightweight cryptography – Part 2: Block ciphers

なお，ここで提案しているアルゴリズム仕様は本書の仕様と同一である．

IETF　Internet Draft:
M. Katagi, S. Moriai, "The 128-bit Blockcipher CLEFIA", October 19, 2009. http://tools.ietf.org/html/draft-katagi-clefia-00

上記 Internet Draft は 2010 年 4 月 22 日に Expire するが，順次更新していく予定である．

5.1.2　製品・システム等での採用実績

2010 年 1 月現在，公開可能な情報はありません．最新の情報については，応募担当者までお問い合わせ下さい．

5.2　推奨用途

CLEFIA は，鍵長 128 ビット，192 ビット，256 ビットに対応した 128 ビットブロック暗号であり，ソフトウェア実装，ハードウェア実装いずれにおいても高速に実装可能であることから，高い安全性と高い実装性能が求められる電子政府システムのあらゆる分野での利用に適している．

また，ハードウェア実装における小型実装性能に優れているため，高い実装制約のある環境での製品やシステムにおける利用にも適している．

参考文献

[1] E. Biham, O. Dunkelman, and N. Keller, "Related-Key Impossible Differential Attacks on 8-Round AES-192." in *Topics in Cryptology – CT-RSA 2006, The Cryptographers' Track* (D. Pointcheval, ed.), no. 3860 in LNCS, pp. 21–33, Springer-Verlag, 2006.

[2] A. Biryukov and D. Khovratovich, "Related-Key Cryptanalysis of the Full AES-192 and AES-256." in *Advances in Cryptology – ASIACRYPT 2009* (M. Matsui, ed.), no. 5912 in LNCS, pp. 1–18, Springer-Verlag, 2009.

[3] A. Biryukov, D. Khovratovich, and I. Nikolić, "Distinguisher and Related-Key Attack on the Full AES-256." in *Advances in Cryptology – CRYPTO 2009* (S. Halevi, ed.), no. 5677 in LNCS, pp. 231–249, Springer-Verlag, 2009.

[4] N. Courtois and J. Pieprzyk, "Cryptanalysis of block ciphers with overdefined systems of equations." in *Proceedings of ASIACRYPT'02* (Y. Zheng, ed.), no. 2501 in LNCS, pp. 267–287, Springer-Verlag, 2002.

[5] J. Daemen and V. Rijmen, *The Design of Rijndael: AES – The Advanced Encryption Standard (Information Security and Cryptography).* Springer, 2002.

[6] A. Rudra, P. K. Dubey, C. S. Jutla, V. Kumar, J. R. Rao, and P. Rohatgi, "Efficient Rijndael encryption implementation with composite field arithmetic." in *Proceedings of Cryptographic Hardware and Embedded Systems – CHES 2001* (Ç. Koç, D. Naccache, and C. Paar, eds.), no. 2162 in LNCS, pp. 171–184, Springer-Verlag, 2001.

[7] T. Shirai and B. Preneel, "On Feistel ciphers using optimal diffusion mappings across multiple rounds." in *Proceedings of ASIACRYPT'04* (P. J. Lee, ed.), no. 3329 in LNCS, pp. 1–15, Springer-Verlag, 2004.

[8] T. Shirai and K. Shibutani, "On Feistel structures using a diffusion switching mechanism." in *Proceedings of Fast Software Encryption – FSE'06* (M. Robshaw, ed.), no. 4047 in LNCS, pp. 41–56, Springer-Verlag, 2006.

[9] T. Shirai, K. Shibutani, T. Akishita, S. Moriai, and T. Iwata, "Hardware Implementations of the 128-bit Blockcipher CLEFIA." in *ISEC Technical Report – ISEC2007-49 (in Japanese)*, 2007.

[10] T. Sugawara, N. Homma, T. Aoki, and A. Satoh, "ASIC Implementations of the 128-bit Block Cipher CLEFIA." in *Proceedings of Computer Security Symposium 2007 – CSS2007 (in Japanese)*, pp. 175–180, 2007.

[11] T. Sugawara, N. Homma, T. Aoki, and A. Satoh, "ASIC Performance Comparison for the ISO Standard Block Ciphers." in *Proceedings of the 2nd Joint Workshop on Information security – JWIS2007*, pp. 485–498, 2007.

[12] Y. Zheng, T. Matsumoto, and H. Imai, "On the Construction of Block Ciphers Provably Secure and Not Relying on Any Unproved Hypotheses." in *Proceedings of CRYPTO 89* (G. Brassard, ed.), no. 435 in LNCS, pp. 461–480, Springer-Verlag, 1989.

著作権について