128ビットブロック暗号 *Camellia* アルゴリズム仕様書

青木 和麻呂 †, 市川 哲也 ‡, 神田 雅透 †,
松井 充 ‡, 盛合 志帆 †, 中嶋 純子 ‡, 時田 俊雄 ‡

† 日本電信電話株式会社　‡ 三菱電機株式会社

第 1.0 版:　2000 年 7 月 12 日
(翻訳:　　2000 年 9 月 21 日)
第 1.1 版:　2001 年 8 月 30 日
第 2.0 版:　2001 年 9 月 26 日

目 次

1　はじめに **4**

2　表記方法について **4**
2.1　基数 4
2.2　記号 4
2.3　演算子 4
2.4　データフォーマット 5

3　*Camellia* の構造 **6**
3.1　関数と変数の一覧 6
3.2　暗号化処理 7
3.2.1　128 ビット鍵での暗号化 7
3.2.2　192 ビット鍵と 256 ビット鍵での暗号化 7
3.3　復号処理 8
3.3.1　128 ビット鍵での復号 8
3.3.2　192 ビット鍵と 256 ビット鍵での復号 9
3.4　鍵スケジュール 9

4　*Camellia* の構成要素 **12**
4.1　F 関数 12

4.2 FL 関数 . . . 12
4.3 FL^{-1} 関数 . . . 12
4.4 S 関数 . . . 12
4.5 置換表 . . . 13
4.6 P 関数 . . . 17

A $Camellia$ アルゴリズムの構造図 18

B テストデータ 24

C ソフトウエアによる実装手法 24
C.1 準備部 . . . 24
C.1.1 全ての副鍵の記憶 . . . 24
C.1.2 副鍵の生成順序 . . . 25
C.1.3 鍵スケジュール中の排他的論理和による相殺 . . . 25
C.1.4 K_L, K_R, K_A, K_B の循環ビット数 . . . 25
C.1.5 k_{11} と k_{12} から kl_5 と kl_6 の生成 . . . 25
C.1.6 動的副鍵生成 . . . 25
C.1.7 128 ビット鍵と 192・256 ビット鍵 . . . 25
C.1.8 **Q** 上の元をどう循環シフトするか . . . 26
C.1.9 F 関数 . . . 26
C.1.10 鍵依存関数 . . . 26
C.2 データ撹拌部 . . . 26
C.2.1 Endian 変換 . . . 26
C.2.2 Little endian 解釈の 1 ビット循環シフト . . . 27
C.2.3 初期及び最終排他的論理和 (whitening) . . . 27
C.2.4 副鍵との排他的論理和 . . . 28
C.2.5 S 関数 . . . 28
C.2.6 P 関数 . . . 28
C.2.7 換字置換 . . . 29
C.2.8 置換表の引数生成 . . . 32
C.3 一般的な注意事項 . . . 32

D 設計方針 33

E 設計基準 34
E.1 F 関数 . . . 34
E.2 P 関数 . . . 34
E.3 置換表 . . . 34
E.4 FL 関数と FL^{-1} 関数 . . . 35
E.5 鍵スケジュール . . . 35

F バージョン情報 **36**

G オブジェクト 識別子 **37**

H 利用実績・推奨用途など **37**

1 はじめに

本稿では共通鍵暗号アルゴリズム *Camellia* の詳細仕様について記載する。Camellia は、ブロック長が 128 ビット、鍵長が 128 または 192 または 256 ビットのいずれかを利用できる。

2 表記方法について

2.1 基数

先頭に **0x** の付いた数字は 16 進数である。

2.2 記号

本稿では、以下に示す記号を使用する。

- $\mathbf{B}$ は 8 ビット (byte) の値全体から構成されるベクトル空間を意味する。すなわち $\mathbf{B} := \mathrm{GF}(2)^8$ である。
- $\mathbf{W}$ は 32 ビット (word) の値全体から構成されるベクトル空間を意味する。すなわち $\mathbf{W} := \mathbf{B}^4$ である。
- $\mathbf{L}$ は 64 ビット (double word) の値全体から構成されるベクトル空間を意味する。すなわち $\mathbf{L} := \mathbf{B}^8$ である。
- $\mathbf{Q}$ は 128 ビット (quad word) の値全体から構成されるベクトル空間を意味する。すなわち $\mathbf{Q} := \mathbf{B}^{16}$ である。
- $x_{(n)}$ のように変数の末尾に $_{(n)}$ が付いている場合、そのビット長が n ビットであることを示す。
- x_L のように変数の末尾に $_L$ が付いている場合、それが x の左側半分であることを意味する。
- x_R のように変数の末尾に $_R$ が付いている場合、それが x の右側半分であることを意味する。

末尾の $_{(n)}$ は、その意味が曖昧でない場合、省略されることがある。「左」と「右」については、第 2.4 章に具体的な例が示されるので参照されたい。

2.3 演算子

$\oplus$	排他的論理和
$\|\|$	結合
$\lll_n$	n ビット左循環シフト
$\cap$	論理積
$\cup$	論理和
$\overline{x}$	x の 1 の補数

2.4 データフォーマット

ビットおよびバイトの並び方は big endian とする。128 ビットデータの $Q_{(128)}$ が、2 個の 64 ビットデータ $L_{i(64)}$ $(i = 1, 2)$、4 個の 32 ビットデータ $W_{i(32)}$ $(i = 1, 2, 3, 4)$、16 個の 8 ビットデータ $B_{i(8)}$ $(i = 1, 2, \ldots, 16)$、あるいは 128 個の 1 ビットデータ $E_{i(1)}$ $(i = 1, 2, \ldots, 128)$ で、それぞれ次のように構成される。

$$
\begin{aligned}
Q_{(128)} &= L_{1(64)}||L_{2(64)} \\
&= W_{1(32)}||W_{2(32)}||W_{3(32)}||W_{4(32)} \\
&= B_{1(8)}||B_{2(8)}||B_{3(8)}||B_{4(8)}||\cdots\cdots||B_{13(8)}||B_{14(8)}||B_{15(8)}||B_{16(8)} \\
&= E_{1(1)}||E_{2(1)}||E_{3(1)}||E_{4(1)}||\cdots\cdots\cdots\cdots\cdots\cdots||E_{125(1)}||E_{126(1)}||E_{127(1)}||E_{128(1)}
\end{aligned}
$$

数値例:

$$
\begin{aligned}
Q_{(128)} &= \texttt{0x0123456789ABCDEF0011223344556677}_{(128)} \\
L_{1(64)} &= Q_{L(64)} = \texttt{0x0123456789ABCDEF}_{(64)} \\
L_{2(64)} &= Q_{R(64)} = \texttt{0x0011223344556677}_{(64)} \\
W_{1(32)} &= L_{1L(32)} = \texttt{0x01234567}_{(32)} \\
W_{2(32)} &= L_{1R(32)} = \texttt{0x89ABCDEF}_{(32)} \\
W_{3(32)} &= L_{2L(32)} = \texttt{0x00112233}_{(32)} \\
W_{4(32)} &= L_{2R(32)} = \texttt{0x44556677}_{(32)} \\
B_{1(8)} &= \texttt{0x01}_{(8)}, \quad B_{2(8)} = \texttt{0x23}_{(8)}, \quad B_{3(8)} = \texttt{0x45}_{(8)}, \quad B_{4(8)} = \texttt{0x67}_{(8)}, \\
B_{5(8)} &= \texttt{0x89}_{(8)}, \quad B_{6(8)} = \texttt{0xAB}_{(8)}, \quad B_{7(8)} = \texttt{0xCD}_{(8)}, \quad B_{8(8)} = \texttt{0xEF}_{(8)}, \\
B_{9(8)} &= \texttt{0x00}_{(8)}, \quad B_{10(8)} = \texttt{0x11}_{(8)}, \quad B_{11(8)} = \texttt{0x22}_{(8)}, \quad B_{12(8)} = \texttt{0x33}_{(8)}, \\
B_{13(8)} &= \texttt{0x44}_{(8)}, \quad B_{14(8)} = \texttt{0x55}_{(8)}, \quad B_{15(8)} = \texttt{0x66}_{(8)}, \quad B_{16(8)} = \texttt{0x77}_{(8)}, \\
E_{1(1)} &= \texttt{0}_{(1)}, \quad E_{2(1)} = \texttt{0}_{(1)}, \quad E_{3(1)} = \texttt{0}_{(1)}, \quad E_{4(1)} = \texttt{0}_{(1)}, \\
E_{5(1)} &= \texttt{0}_{(1)}, \quad E_{6(1)} = \texttt{0}_{(1)}, \quad E_{7(1)} = \texttt{0}_{(1)}, \quad E_{8(1)} = \texttt{1}_{(1)}, \\
&\vdots \\
E_{121(1)} &= \texttt{0}_{(1)}, \quad E_{122(1)} = \texttt{1}_{(1)}, \quad E_{123(1)} = \texttt{1}_{(1)}, \quad E_{124(1)} = \texttt{1}_{(1)}, \\
E_{125(1)} &= \texttt{0}_{(1)}, \quad E_{126(1)} = \texttt{1}_{(1)}, \quad E_{127(1)} = \texttt{1}_{(1)}, \quad E_{128(1)} = \texttt{1}_{(1)}.
\end{aligned}
$$

$$
\begin{aligned}
Q_{(128)} \lll_1 &= E_{2(1)}||E_{3(1)}||E_{4(1)}||E_{5(1)}||\cdots\cdots\cdots\cdots\cdots\cdots||E_{125(1)}||E_{126(1)}||E_{127(1)}||E_{128(1)}||E_{1(1)} \\
&= \texttt{0x02468ACF13579BDE0022446688AACCEE}_{(128)}
\end{aligned}
$$

3 *Camellia* の構造

3.1 関数と変数の一覧

$M_{(128)}$	平文
$C_{(128)}$	暗号文
K	128、192、または 256 ビット長の秘密鍵
$kw_{t(64)}, k_{u(64)}, kl_{v(64)}$	副鍵 128 ビット鍵の場合: $(t = 1, 2, 3, 4)$ $(u = 1, 2, \ldots, 18)$ $(v = 1, 2, 3, 4)$ 192 または 256 ビット鍵の場合: $(t = 1, 2, 3, 4)$ $(u = 1, 2, \ldots, 24)$ $(v = 1, 2, \ldots, 6)$
$Y_{(64)} = F(X_{(64)},\ k_{(64)})$	F 関数は、64 ビットデータの $X_{(64)}$、および副鍵 $k_{(64)}$ を入力とし、64 ビットデータの $Y_{(64)}$ を出力とする。
$Y_{(64)} = FL(X_{(64)}, k_{(64)})$	FL 関数は、64 ビットデータの $X_{(64)}$、および副鍵 $k_{(64)}$ を入力とし、64 ビットデータの $Y_{(64)}$ を出力とする。
$Y_{(64)} = FL^{-1}(X_{(64)}, k_{(64)})$	FL^{-1} 関数は、64 ビットデータの $X_{(64)}$、および副鍵 $k_{(64)}$ を入力とし、64 ビットデータの $Y_{(64)}$ を出力とする。
$Y_{(64)} = S(X_{(64)})$	S 関数は、64 ビットデータの $X_{(64)}$ を入力とし、64 ビットデータの $Y_{(64)}$ を出力とする。
$Y_{(64)} = P(X_{(64)})$	P 関数は、64 ビットの $X_{(64)}$ を入力とし、64 ビットの $Y_{(64)}$ を出力とする。
$y_{(8)} = s_i(x_{(8)})$	置換表の入出力は 8 ビットである。 $(i = 1, 2, 3, 4)$

3.2 暗号化処理

3.2.1 128 ビット鍵での暗号化

128 ビット鍵の場合の暗号化処理を図 1 に示す。データ攪拌部は、18 段の Feistel 構造と FL/FL^{-1} 関数により構成されている。FL/FL^{-1} 関数は 2 層あり、第 6 段と第 12 段の直後に挿入されている。また、第 1 段の直前と最終段の直後において、初期及び最終排他的論理和 (whitening) がそれぞれ行われる。鍵スケジュール部では秘密鍵 K から、副鍵 $kw_{t(64)}$ $(t = 1, 2, 3, 4)$, $k_{u(64)}$ $(u = 1, 2, \ldots, 18)$, $kl_{v(64)}$ $(v = 1, 2, 3, 4)$ を生成する。鍵スケジュール部の詳細については、第 3.4 章を参照のこと。

データ攪拌部では、まず平文 $M_{(128)}$ と $kw_{1(64)}||kw_{2(64)}$ とを排他的論理和した後、それを $L_{0(64)}$ と $R_{0(64)}$ とに等分する。すなわち、$M_{(128)} \oplus (kw_{1(64)}||kw_{2(64)}) = L_{0(64)}||R_{0(64)}$ とする。そして、以下の演算を $r = 1$ から 18 まで実行する。ただし、$r = 6, 12$ を除く。

$$\begin{aligned} L_r &= R_{r-1} \oplus F(L_{r-1}, k_r) \\ R_r &= L_{r-1} \end{aligned}$$

$r = 6, 12$ の場合は、以下を実行する。

$$\begin{aligned} L'_r &= R_{r-1} \oplus F(L_{r-1}, k_r) \\ R'_r &= L_{r-1} \\ L_r &= FL(L'_r, kl_{2r/6-1}) \\ R_r &= FL^{-1}(R'_r, kl_{2r/6}) \end{aligned}$$

最後に、$R_{18(64)}$ と $L_{18(64)}$ とを結合して得られる 128 ビット長のデータと、副鍵 $kw_{3(64)}||kw_{4(64)}$ との排他的論理和をとったものが暗号文である。すなわち、$C_{(128)} = (R_{18(64)}||L_{18(64)}) \oplus (kw_{3(64)}||kw_{4(64)})$ である。

3.2.2 192 ビット鍵と 256 ビット鍵での暗号化

192 ビット鍵または 256 ビット鍵の場合の暗号化処理を図 2 に示す。データ攪拌部は、24 段の Feistel 構造と FL/FL^{-1} 関数とから構成されている。FL/FL^{-1} 関数は 3 層あり、第 6 段、第 12 段、第 18 段の後に挿入される。また、第 1 段の直前と最終段の直後において、初期及び最終排他的論理和 (whitening) がそれぞれ行われる。鍵スケジュール部では秘密鍵 K から、副鍵 $kw_{t(64)}$ $(t = 1, 2, 3, 4)$, $k_{u(64)}$ $(u = 1, 2, \ldots, 24)$, $kl_{v(64)}$ $(v = 1, 2, \ldots, 6)$ を生成する。

データ攪拌部では、まず平文 $M_{(128)}$ と副鍵 $kw_{1(64)}||kw_{2(64)}$ とを排他的論理和した後、それを $L_{0(64)}$ と $R_{0(64)}$ とに等分する。すなわち $M_{(128)} \oplus (kw_{1(64)}||kw_{2(64)}) = L_{0(64)}||R_{0(64)}$ とする。次に以下の演算を $r = 1$ から 24 まで実行する。ただし、$r = 6, 12, 18$ を除く。

$$\begin{aligned} L_r &= R_{r-1} \oplus F(L_{r-1}, k_r) \\ R_r &= L_{r-1} \end{aligned}$$

$r = 6, 12, 18$ の場合は、以下を実行する。

$$\begin{aligned} L'_r &= R_{r-1} \oplus F(L_{r-1}, k_r) \\ R'_r &= L_{r-1} \\ L_r &= FL(L'_r, kl_{2r/6-1}) \\ R_r &= FL^{-1}(R'_r, kl_{2r/6}) \end{aligned}$$

最後に、$R_{24(64)}$ と $L_{24(64)}$ とを結合して得られる 128 ビットデータと、副鍵 $kw_{3(64)}||kw_{4(64)}$ とを排他的論理和した結果が暗号文である。すなわち、$C_{(128)} = (R_{24(64)}||L_{24(64)}) \oplus (kw_{3(64)}||kw_{4(64)})$ である。

F 関数 と FL/FL^{-1} 関数の詳細については、第 4 章を参照のこと。

3.3 復号処理

3.3.1 128 ビット鍵での復号

Camellia の復号は、副鍵の順番を逆順にすれば暗号化と同様の処理で行うことができる。

128 ビット鍵の場合の復号処理を図 3 に示す。データ攪拌部は、18 段の Feistel 構造と FL/FL^{-1} 関数により構成されている。FL/FL^{-1} 関数は 2 層あり、第 6 段と第 12 段の直後に挿入されている。また、第 1 段の直前と最終段の直後において、初期及び最終排他的論理和 (whitening) がそれぞれ行われる。鍵スケジュール部では秘密鍵 K から、副鍵 $kw_{t(64)}$ $(t = 1, 2, 3, 4)$, $k_{u(64)}$ $(u = 1, 2, \ldots, 18)$, $kl_{v(64)}$ $(v = 1, 2, 3, 4)$ を生成する。鍵スケジュール部の詳細については、第 3.4 章を参照のこと。

データ攪拌部では、まず暗号文 $C_{(128)}$ と副鍵 $kw_{3(64)}||kw_{4(64)}$ とを排他的論理和した後、それを $R_{18(64)}$ と $L_{18(64)}$ とに等分する。すなわち $C_{(128)} \oplus (kw_{3(64)}||kw_{4(64)}) = R_{18(64)}||L_{18(64)}$ とする。次に以下の演算を $r = 18$ から 1 まで降順に実行する。ただし、$r = 13, 7$ を除く。

$$\begin{aligned} R_{r-1} &= L_r \oplus F(R_r, k_r) \\ L_{r-1} &= R_r \end{aligned}$$

$r = 13, 7$ の場合は、以下を実行する。

$$\begin{aligned} R'_{r-1} &= L_r \oplus F(R_r, k_r) \\ L'_{r-1} &= R_r \\ R_{r-1} &= FL(R'_{r-1}, kl_{2(r-1)/6}) \\ L_{r-1} &= FL^{-1}(L'_{r-1}, kl_{2(r-1)/6-1}) \end{aligned}$$

最後に、$L_{0(64)}$ と $R_{0(64)}$ とを結合して得られる 128 ビットのデータと、副鍵 $kw_{1(64)}||kw_{2(64)}$ との排他的論理和した結果が平文である。すなわち、$M_{(128)} = (L_{0(64)}||R_{0(64)}) \oplus (kw_{1(64)}||kw_{2(64)})$ である。

3.3.2 192ビット鍵と256ビット鍵での復号

192ビット鍵または256ビット鍵の場合の復号処理を図4に示す。データ攪拌部は、24段のFeistel構造と FL/FL^{-1} 関数により構成されている。FL/FL^{-1} 関数は3層あり、第6段、第12段、第18段の後に挿入されている。また、第1段の直前と最終段の直後において、初期及び最終排他的論理和(whitening)がそれぞれ行われる。鍵スケジュール部では秘密鍵 K から、副鍵 $kw_{t(64)}$ $(t = 1, 2, 3, 4)$, $k_{u(64)}$ $(u = 1, 2, \ldots, 24)$, $kl_{v(64)}$ $(v = 1, 2, 3, 4)$ $(v = 1, 2, \ldots, 6)$ を生成する。

データ攪拌部では、まず暗号文 $C_{(128)}$ と副鍵 $kw_{3(64)}||kw_{4(64)}$ とを排他的論理和した後、それを $R_{24(64)}$ と $L_{24(64)}$ とに等分する。すなわち、$C_{(128)} \oplus (kw_{3(64)}||kw_{4(64)}) = R_{24(64)}||L_{24(64)}$ とする。そして、以下の演算を $r = 24$ から1まで降順に実行する。ただし、$r = 19, 13, 7$ を除く。

$$\begin{aligned} R_{r-1} &= L_r \oplus F(R_r, k_r) \\ L_{r-1} &= R_r \end{aligned}$$

$r = 19, 13, 7$ の場合は、以下を実行する。

$$\begin{aligned} R'_{r-1} &= L_r \oplus F(R_r, k_r) \\ L'_{r-1} &= R_r \\ R_{r-1} &= FL(R'_{r-1}, kl_{2(r-1)/6}) \\ L_{r-1} &= FL^{-1}(L'_{r-1}, kl_{2(r-1)/6-1}) \end{aligned}$$

最後に、$L_{0(64)}$ と $R_{0(64)}$ とを結合して得られる128ビットのデータと、副鍵 $kw_{1(64)}||kw_{2(64)}$ とを排他的論理和した結果が平文である。すなわち、$M_{(128)} = (L_{0(64)}||R_{0(64)}) \oplus (kw_{1(64)}||kw_{2(64)})$ である。

3.4 鍵スケジュール

Camelliaの鍵スケジュール部では、2つの128ビットデータ $K_{L(128)}$, $K_{R(128)}$ および、4つの64ビットデータ $K_{LL(64)}$, $K_{LR(64)}$, $K_{RL(64)}$, $K_{RR(64)}$ を用いる。これらは以下に示す関係を満たすように定義される:

$$\begin{aligned} K_{(128)} &= K_{L(128)}, & K_{R(128)} &= 0; & &\text{(128ビット鍵)} \\ K_{(192)} &= K_{L(128)}||K_{RL(64)}, & K_{RR(64)} &= \overline{K_{RL(64)}}; & &\text{(192ビット鍵)} \\ K_{(256)} &= K_{L(128)}||K_{R(128)}; & & & &\text{(256ビット鍵)} \end{aligned}$$

$$\begin{aligned} K_{L(128)} &= K_{LL(64)}||K_{LR(64)}, \\ K_{R(128)} &= K_{RL(64)}||K_{RR(64)}; \end{aligned} \qquad \text{(すべての鍵)}$$

これらの値を用いて、図 8 に示すように 2 つの 128 ビットデータ $K_{A(128)}$ および $K_{B(128)}$ を生成する。ただし $K_{B(128)}$ の方は 192 あるいは 256 ビット鍵の場合にのみ使用する。まず、$K = K_{L(128)}$ と $K_{R(128)}$ とを排他的論理和した後、2 段分の「暗号化」を行う。このとき、定数 $\Sigma_{1(64)}, \Sigma_{2(64)}$ を「鍵」として使用する。次にその結果は $K_{L(128)}$ と排他的論理和され、今度は $\Sigma_{3(64)}, \Sigma_{4(64)}$ を使用し再び 2 段分の暗号化処理を行う。この結果得られる値が $K_{A(128)}$ である。最後に $K_{A(128)}$ は $K_{R(128)}$ と排他的論理和され、$\Sigma_{5(64)}, \Sigma_{6(64)}$ を用いて、2 段分の暗号化処理を行う。この結果得られる値が $K_{B(128)}$ である。Σ_i は、2 から始まる i 番目の素数の、2 乗根の 16 進表示の小数第 2 位から 16 桁取り出したものである。これら定数の値を表 1 に示す。

副鍵 $kw_{t(64)}, k_{u(64)}, kl_{v(64)}$ は、$K_{L(128)}, K_{R(128)}, K_{A(128)}, K_{B(128)}$ を循環シフトさせた値の左、あるいは右半分の値となっている。その詳細については、表 2、 3 を参照のこと。

互換性 $K_{RR(64)} = \overline{K_{RL(64)}}$ とすれば、256 ビット鍵と 192 ビット鍵での暗号化、復号処理には互換性が保たれる。128 ビット鍵の場合は、192 ビット鍵または 256 ビット鍵とはラウンド関数の段数が異なるため互換性はない。

表 1: 鍵スケジュールに用いる定数

$\Sigma_{1(64)}$	0xA09E667F3BCC908B
$\Sigma_{2(64)}$	0xB67AE8584CAA73B2
$\Sigma_{3(64)}$	0xC6EF372FE94F82BE
$\Sigma_{4(64)}$	0x54FF53A5F1D36F1C
$\Sigma_{5(64)}$	0x10E527FADE682D1D
$\Sigma_{6(64)}$	0xB05688C2B3E6C1FD

表 2: 128 ビット鍵における副鍵

	副鍵	値
初期	$kw_{1(64)}$	$(K_L \lll_0)_{L(64)}$
排他的論理和	$kw_{2(64)}$	$(K_L \lll_0)_{R(64)}$
F (第 1 段)	$k_{1(64)}$	$(K_A \lll_0)_{L(64)}$
F (第 2 段)	$k_{2(64)}$	$(K_A \lll_0)_{R(64)}$
F (第 3 段)	$k_{3(64)}$	$(K_L \lll_{15})_{L(64)}$
F (第 4 段)	$k_{4(64)}$	$(K_L \lll_{15})_{R(64)}$
F (第 5 段)	$k_{5(64)}$	$(K_A \lll_{15})_{L(64)}$
F (第 6 段)	$k_{6(64)}$	$(K_A \lll_{15})_{R(64)}$
FL	$kl_{1(64)}$	$(K_A \lll_{30})_{L(64)}$
FL^{-1}	$kl_{2(64)}$	$(K_A \lll_{30})_{R(64)}$
F (第 7 段)	$k_{7(64)}$	$(K_L \lll_{45})_{L(64)}$
F (第 8 段)	$k_{8(64)}$	$(K_L \lll_{45})_{R(64)}$
F (第 9 段)	$k_{9(64)}$	$(K_A \lll_{45})_{L(64)}$
F (第 10 段)	$k_{10(64)}$	$(K_L \lll_{60})_{R(64)}$
F (第 11 段)	$k_{11(64)}$	$(K_A \lll_{60})_{L(64)}$
F (第 12 段)	$k_{12(64)}$	$(K_A \lll_{60})_{R(64)}$
FL	$kl_{3(64)}$	$(K_L \lll_{77})_{L(64)}$
FL^{-1}	$kl_{4(64)}$	$(K_L \lll_{77})_{R(64)}$
F (第 13 段)	$k_{13(64)}$	$(K_L \lll_{94})_{L(64)}$
F (第 14 段)	$k_{14(64)}$	$(K_L \lll_{94})_{R(64)}$
F (第 15 段)	$k_{15(64)}$	$(K_A \lll_{94})_{L(64)}$
F (第 16 段)	$k_{16(64)}$	$(K_A \lll_{94})_{R(64)}$
F (第 17 段)	$k_{17(64)}$	$(K_L \lll_{111})_{L(64)}$
F (第 18 段)	$k_{18(64)}$	$(K_L \lll_{111})_{R(64)}$
最終	$kw_{3(64)}$	$(K_A \lll_{111})_{L(64)}$
排他的論理和	$kw_{4(64)}$	$(K_A \lll_{111})_{R(64)}$

表 3: 192, 256 ビット鍵における副鍵

	副鍵	値
初期	$kw_{1(64)}$	$(K_L \lll_0)_{L(64)}$
排他的論理和	$kw_{2(64)}$	$(K_L \lll_0)_{R(64)}$
F (第 1 段)	$k_{1(64)}$	$(K_B \lll_0)_{L(64)}$
F (第 2 段)	$k_{2(64)}$	$(K_B \lll_0)_{R(64)}$
F (第 3 段)	$k_{3(64)}$	$(K_R \lll_{15})_{L(64)}$
F (第 4 段)	$k_{4(64)}$	$(K_R \lll_{15})_{R(64)}$
F (第 5 段)	$k_{5(64)}$	$(K_A \lll_{15})_{L(64)}$
F (第 6 段)	$k_{6(64)}$	$(K_A \lll_{15})_{R(64)}$
FL	$kl_{1(64)}$	$(K_R \lll_{30})_{L(64)}$
FL^{-1}	$kl_{2(64)}$	$(K_R \lll_{30})_{R(64)}$
F (第 7 段)	$k_{7(64)}$	$(K_B \lll_{30})_{L(64)}$
F (第 8 段)	$k_{8(64)}$	$(K_B \lll_{30})_{R(64)}$
F (第 9 段)	$k_{9(64)}$	$(K_L \lll_{45})_{L(64)}$
F (第 10 段)	$k_{10(64)}$	$(K_L \lll_{45})_{R(64)}$
F (第 11 段)	$k_{11(64)}$	$(K_A \lll_{45})_{L(64)}$
F (第 12 段)	$k_{12(64)}$	$(K_A \lll_{45})_{R(64)}$
FL	$kl_{3(64)}$	$(K_L \lll_{60})_{L(64)}$
FL^{-1}	$kl_{4(64)}$	$(K_L \lll_{60})_{R(64)}$
F (第 13 段)	$k_{13(64)}$	$(K_R \lll_{60})_{L(64)}$
F (第 14 段)	$k_{14(64)}$	$(K_R \lll_{60})_{R(64)}$
F (第 15 段)	$k_{15(64)}$	$(K_B \lll_{60})_{L(64)}$
F (第 16 段)	$k_{16(64)}$	$(K_B \lll_{60})_{R(64)}$
F (第 17 段)	$k_{17(64)}$	$(K_L \lll_{77})_{L(64)}$
F (第 18 段)	$k_{18(64)}$	$(K_L \lll_{77})_{R(64)}$
FL	$kl_{5(64)}$	$(K_A \lll_{77})_{L(64)}$
FL^{-1}	$kl_{6(64)}$	$(K_A \lll_{77})_{R(64)}$
F (第 19 段)	$k_{19(64)}$	$(K_R \lll_{94})_{L(64)}$
F (第 20 段)	$k_{20(64)}$	$(K_R \lll_{94})_{R(64)}$
F (第 21 段)	$k_{21(64)}$	$(K_A \lll_{94})_{L(64)}$
F (第 22 段)	$k_{22(64)}$	$(K_A \lll_{94})_{R(64)}$
F (第 23 段)	$k_{23(64)}$	$(K_L \lll_{111})_{L(64)}$
F (第 24 段)	$k_{24(64)}$	$(K_L \lll_{111})_{R(64)}$
最終	$kw_{3(64)}$	$(K_B \lll_{111})_{L(64)}$
排他的論理和	$kw_{4(64)}$	$(K_B \lll_{111})_{R(64)}$

4 *Camellia* の構成要素

4.1 *F* 関数

F 関数は図 5 に示され、その定義は次の通りである。

$$\begin{aligned} F : \mathbf{L} \times \mathbf{L} &\longrightarrow \mathbf{L} \\ (X_{(64)}, k_{(64)}) &\longmapsto Y_{(64)} = P(S(X_{(64)} \oplus k_{(64)})) \end{aligned}$$

S 関数と P 関数については、それぞれ第 4.4 章、第 4.6 章を参照のこと。

4.2 *FL* 関数

FL 関数は図 6 に示され、その定義は次の通りである。

$$\begin{aligned} FL : \mathbf{L} \times \mathbf{L} &\longrightarrow \mathbf{L} \\ (X_{L(32)}||X_{R(32)}, kl_{L(32)}||kl_{R(32)}) &\longmapsto Y_{L(32)}||Y_{R(32)} \end{aligned}$$

ここで、

$$\begin{aligned} Y_{R(32)} &= ((X_{L(32)} \cap kl_{L(32)}) \lll_1) \oplus X_{R(32)} \\ Y_{L(32)} &= (Y_{R(32)} \cup kl_{R(32)}) \oplus X_{L(32)} \end{aligned}$$

4.3 *FL*$^{-1}$ 関数

FL^{-1} 関数は図 7 に示され、その定義は次の通りである。

$$\begin{aligned} FL^{-1} : \mathbf{L} \times \mathbf{L} &\longrightarrow \mathbf{L} \\ (Y_{L(32)}||Y_{R(32)}, kl_{L(32)}||kl_{R(32)}) &\longmapsto X_{L(32)}||X_{R(32)} \end{aligned}$$

ここで、

$$\begin{aligned} X_{L(32)} &= (Y_{R(32)} \cup kl_{R(32)}) \oplus Y_{L(32)} \\ X_{R(32)} &= ((X_{L(32)} \cap kl_{L(32)}) \lll_1) \oplus Y_{R(32)} \end{aligned}$$

4.4 *S* 関数

S 関数は F 関数の部分関数であり、その定義は次の通りである。

$$\begin{aligned} S : \mathbf{L} &\longrightarrow \mathbf{L} \\ l_{1(8)}||l_{2(8)}||l_{3(8)}||l_{4(8)}||l_{5(8)}||l_{6(8)}||l_{7(8)}||l_{8(8)} &\longmapsto l'_{1(8)}||l'_{2(8)}||l'_{3(8)}||l'_{4(8)}||l'_{5(8)}||l'_{6(8)}||l'_{7(8)}||l'_{8(8)} \end{aligned}$$

$$
\begin{aligned}
l'_{1(8)} &= s_1(l_{1(8)}),\\
l'_{2(8)} &= s_2(l_{2(8)}),\\
l'_{3(8)} &= s_3(l_{3(8)}),\\
l'_{4(8)} &= s_4(l_{4(8)}),\\
l'_{5(8)} &= s_2(l_{5(8)}),\\
l'_{6(8)} &= s_3(l_{6(8)}),\\
l'_{7(8)} &= s_4(l_{7(8)}),\\
l'_{8(8)} &= s_1(l_{8(8)}),
\end{aligned}
$$

ここで、4 つの置換表、 s_1, s_2, s_3, s_4 については、第 4.5 章に記述される。

4.5 置換表

Camellia で用いる 4 つの置換表は、GF(2^8) 上の逆元算と GF$(2)^8$ 上のアフィン変換との合成写像である。置換表の仕様を、表 4、 5、 6 および 7 に示す。また、置換表の代数的表現を次に示す。

$$
\begin{aligned}
s_1 &: \mathbf{B} \longrightarrow \mathbf{B}\\
x_{(8)} &\longmapsto \mathbf{h}(\mathbf{g}(\mathbf{f}(\texttt{0xc5} \oplus x_{(8)}))) \oplus \texttt{0x6e}\\
s_2 &: \mathbf{B} \longrightarrow \mathbf{B}\\
x_{(8)} &\longmapsto s_1(x_{(8)}) \lll_1\\
s_3 &: \mathbf{B} \longrightarrow \mathbf{B}\\
x_{(8)} &\longmapsto s_1(x_{(8)}) \ggg_1\\
s_4 &: \mathbf{B} \longrightarrow \mathbf{B}\\
x_{(8)} &\longmapsto s_1(x_{(8)} \lll_1)
\end{aligned}
$$

ここで $\mathbf{f}, \mathbf{g}, \mathbf{h}$ 関数は以下のように与えられる:

$$
\begin{aligned}
&\mathbf{f} : \mathbf{B} \longrightarrow \mathbf{B}\\
&a_{1(1)}||a_{2(1)}||a_{3(1)}||a_{4(1)}||a_{5(1)}||a_{6(1)}||a_{7(1)}||a_{8(1)}\\
&\longmapsto b_{1(1)}||b_{2(1)}||b_{3(1)}||b_{4(1)}||b_{5(1)}||b_{6(1)}||b_{7(1)}||b_{8(1)}
\end{aligned}
$$

ここで、

$$
\begin{aligned}
b_1 &= a_6 \oplus a_2,\\
b_2 &= a_7 \oplus a_1,\\
b_3 &= a_8 \oplus a_5 \oplus a_3,
\end{aligned}
$$

$$
\begin{aligned}
b_4 &= a_8 \oplus a_3, \\
b_5 &= a_7 \oplus a_4, \\
b_6 &= a_5 \oplus a_2, \\
b_7 &= a_8 \oplus a_1, \\
b_8 &= a_6 \oplus a_4.
\end{aligned}
$$

$$
\begin{aligned}
&\mathbf{g} : \mathbf{B} \longrightarrow \mathbf{B} \\
&a_{1(1)}||a_{2(1)}||a_{3(1)}||a_{4(1)}||a_{5(1)}||a_{6(1)}||a_{7(1)}||a_{8(1)} \\
&\longmapsto b_{1(1)}||b_{2(1)}||b_{3(1)}||b_{4(1)}||b_{5(1)}||b_{6(1)}||b_{7(1)}||b_{8(1)}
\end{aligned}
$$

ここで、

$$
\begin{aligned}
&(b_8 + b_7\alpha + b_6\alpha^2 + b_5\alpha^3) + (b_4 + b_3\alpha + b_2\alpha^2 + b_1\alpha^3)\beta \\
&= 1/((a_8 + a_7\alpha + a_6\alpha^2 + a_5\alpha^3) + (a_4 + a_3\alpha + a_2\alpha^2 + a_1\alpha^3)\beta)
\end{aligned}
$$

ここの $\frac{1}{x}$ 写像は、GF(2^8) 上で行われる。ただし、$\frac{1}{0} = 0$ とし、β は、GF(2^8) の要素であり、$\beta^8 + \beta^6 + \beta^5 + \beta^3 + 1 = 0$ を満たすものとする。また、$\alpha = \beta^{238} = \beta^6 + \beta^5 + \beta^3 + \beta^2$ は GF(2^4) の要素であり、$\alpha^4 + \alpha + 1 = 0$ を満たすものとする。

$$
\begin{aligned}
&\mathbf{h} : \mathbf{B} \longrightarrow \mathbf{B} \\
&a_{1(1)}||a_{2(1)}||a_{3(1)}||a_{4(1)}||a_{5(1)}||a_{6(1)}||a_{7(1)}||a_{8(1)} \\
&\longmapsto b_{1(1)}||b_{2(1)}||b_{3(1)}||b_{4(1)}||b_{5(1)}||b_{6(1)}||b_{7(1)}||b_{8(1)}
\end{aligned}
$$

ここで、

$$
\begin{aligned}
b_1 &= a_5 \oplus a_6 \oplus a_2, \\
b_2 &= a_6 \oplus a_2, \\
b_3 &= a_7 \oplus a_4, \\
b_4 &= a_8 \oplus a_2, \\
b_5 &= a_7 \oplus a_3, \\
b_6 &= a_8 \oplus a_1, \\
b_7 &= a_5 \oplus a_1, \\
b_8 &= a_6 \oplus a_3.
\end{aligned}
$$

表 4: 置換表 s_1

このテーブルにおいて、$s_1(0) = 112, s_1(1) = 130, \ldots, s_1(255) = 158$ である。

112	130	44	236	179	39	192	229	228	133	87	53	234	12	174	65
35	239	107	147	69	25	165	33	237	14	79	78	29	101	146	189
134	184	175	143	124	235	31	206	62	48	220	95	94	197	11	26
166	225	57	202	213	71	93	61	217	1	90	214	81	86	108	77
139	13	154	102	251	204	176	45	116	18	43	32	240	177	132	153
223	76	203	194	52	126	118	5	109	183	169	49	209	23	4	215
20	88	58	97	222	27	17	28	50	15	156	22	83	24	242	34
254	68	207	178	195	181	122	145	36	8	232	168	96	252	105	80
170	208	160	125	161	137	98	151	84	91	30	149	224	255	100	210
16	196	0	72	163	247	117	219	138	3	230	218	9	63	221	148
135	92	131	2	205	74	144	51	115	103	246	243	157	127	191	226
82	155	216	38	200	55	198	59	129	150	111	75	19	190	99	46
233	121	167	140	159	110	188	142	41	245	249	182	47	253	180	89
120	152	6	106	231	70	113	186	212	37	171	66	136	162	141	250
114	7	185	85	248	238	172	10	54	73	42	104	60	56	241	164
64	40	211	123	187	201	67	193	21	227	173	244	119	199	128	158

表 5: 置換表 s_2

224	5	88	217	103	78	129	203	201	11	174	106	213	24	93	130
70	223	214	39	138	50	75	66	219	28	158	156	58	202	37	123
13	113	95	31	248	215	62	157	124	96	185	190	188	139	22	52
77	195	114	149	171	142	186	122	179	2	180	173	162	172	216	154
23	26	53	204	247	153	97	90	232	36	86	64	225	99	9	51
191	152	151	133	104	252	236	10	218	111	83	98	163	46	8	175
40	176	116	194	189	54	34	56	100	30	57	44	166	48	229	68
253	136	159	101	135	107	244	35	72	16	209	81	192	249	210	160
85	161	65	250	67	19	196	47	168	182	60	43	193	255	200	165
32	137	0	144	71	239	234	183	21	6	205	181	18	126	187	41
15	184	7	4	155	148	33	102	230	206	237	231	59	254	127	197
164	55	177	76	145	110	141	118	3	45	222	150	38	125	198	92
211	242	79	25	63	220	121	29	82	235	243	109	94	251	105	178
240	49	12	212	207	140	226	117	169	74	87	132	17	69	27	245
228	14	115	170	241	221	89	20	108	146	84	208	120	112	227	73
128	80	167	246	119	147	134	131	42	199	91	233	238	143	1	61

表 6: 置換表 s_3

56	65	22	118	217	147	96	242	114	194	171	154	117	6	87	160
145	247	181	201	162	140	210	144	246	7	167	39	142	178	73	222
67	92	215	199	62	245	143	103	31	24	110	175	47	226	133	13
83	240	156	101	234	163	174	158	236	128	45	107	168	43	54	166
197	134	77	51	253	102	88	150	58	9	149	16	120	216	66	204
239	38	229	97	26	63	59	130	182	219	212	152	232	139	2	235
10	44	29	176	111	141	136	14	25	135	78	11	169	12	121	17
127	34	231	89	225	218	61	200	18	4	116	84	48	126	180	40
85	104	80	190	208	196	49	203	42	173	15	202	112	255	50	105
8	98	0	36	209	251	186	237	69	129	115	109	132	159	238	74
195	46	193	1	230	37	72	153	185	179	123	249	206	191	223	113
41	205	108	19	100	155	99	157	192	75	183	165	137	95	177	23
244	188	211	70	207	55	94	71	148	250	252	91	151	254	90	172
60	76	3	53	243	35	184	93	106	146	213	33	68	81	198	125
57	131	220	170	124	119	86	5	27	164	21	52	30	28	248	82
32	20	233	189	221	228	161	224	138	241	214	122	187	227	64	79

表 7: 置換表 s_4

112	44	179	192	228	87	234	174	35	107	69	165	237	79	29	146
134	175	124	31	62	220	94	11	166	57	213	93	217	90	81	108
139	154	251	176	116	43	240	132	223	203	52	118	109	169	209	4
20	58	222	17	50	156	83	242	254	207	195	122	36	232	96	105
170	160	161	98	84	30	224	100	16	0	163	117	138	230	9	221
135	131	205	144	115	246	157	191	82	216	200	198	129	111	19	99
233	167	159	188	41	249	47	180	120	6	231	113	212	171	136	141
114	185	248	172	54	42	60	241	64	211	187	67	21	173	119	128
130	236	39	229	133	53	12	65	239	147	25	33	14	78	101	189
184	143	235	206	48	95	197	26	225	202	71	61	1	214	86	77
13	102	204	45	18	32	177	153	76	194	126	5	183	49	23	215
88	97	27	28	15	22	24	34	68	178	181	145	8	168	252	80
208	125	137	151	91	149	255	210	196	72	247	219	3	218	63	148
92	2	74	51	103	243	127	226	155	38	55	59	150	75	190	46
121	140	110	142	245	182	253	89	152	106	70	186	37	66	162	250
7	85	238	10	73	104	56	164	40	123	201	193	227	244	199	158

4.6 P 関数

P 関数は F 関数の部分関数であり、その定義は次の通りである。

$$
\begin{array}{rcl}
P : \mathbf{L} & \longrightarrow & \mathbf{L} \\
z_{1(8)}||z_{2(8)}||z_{3(8)}||z_{4(8)}||z_{5(8)}||z_{6(8)}||z_{7(8)}||z_{8(8)} & \longmapsto & z'_{1(8)}||z'_{2(8)}||z'_{3(8)}||z'_{4(8)}||z'_{5(8)}||z'_{6(8)}||z'_{7(8)}||z'_{8(8)},
\end{array}
$$

ここで、

$$
\begin{aligned}
z'_1 &= z_1 \oplus z_3 \oplus z_4 \oplus z_6 \oplus z_7 \oplus z_8 \\
z'_2 &= z_1 \oplus z_2 \oplus z_4 \oplus z_5 \oplus z_7 \oplus z_8 \\
z'_3 &= z_1 \oplus z_2 \oplus z_3 \oplus z_5 \oplus z_6 \oplus z_8 \\
z'_4 &= z_2 \oplus z_3 \oplus z_4 \oplus z_5 \oplus z_6 \oplus z_7 \\
z'_5 &= z_1 \oplus z_2 \oplus z_6 \oplus z_7 \oplus z_8 \\
z'_6 &= z_2 \oplus z_3 \oplus z_5 \oplus z_7 \oplus z_8 \\
z'_7 &= z_3 \oplus z_4 \oplus z_5 \oplus z_6 \oplus z_8 \\
z'_8 &= z_1 \oplus z_4 \oplus z_5 \oplus z_6 \oplus z_7
\end{aligned}
$$

であり、この変換は次のような行列を用いた線形写像で表すことができる。

$$
\begin{pmatrix} z_8 \\ z_7 \\ \vdots \\ z_1 \end{pmatrix} \longmapsto \begin{pmatrix} z'_8 \\ z'_7 \\ \vdots \\ z'_1 \end{pmatrix} = P \begin{pmatrix} z_8 \\ z_7 \\ \vdots \\ z_1 \end{pmatrix}
$$

ここで、

$$
P = \begin{pmatrix}
0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\
1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\
1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\
1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\
1 & 1 & 1 & 0 & 1 & 1 & 0 & 1
\end{pmatrix}
$$

である。

A *Camellia* アルゴリズムの構造図

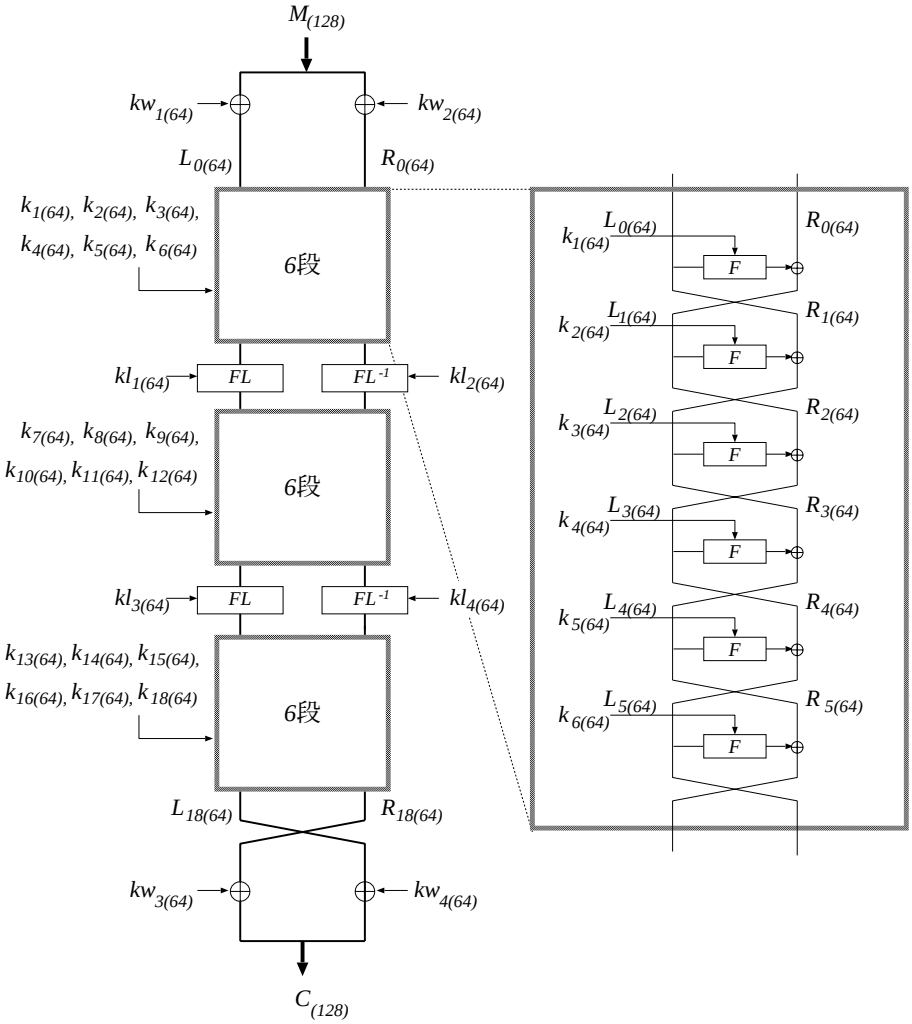

図 1: 128 ビット鍵 Camellia の暗号化処理

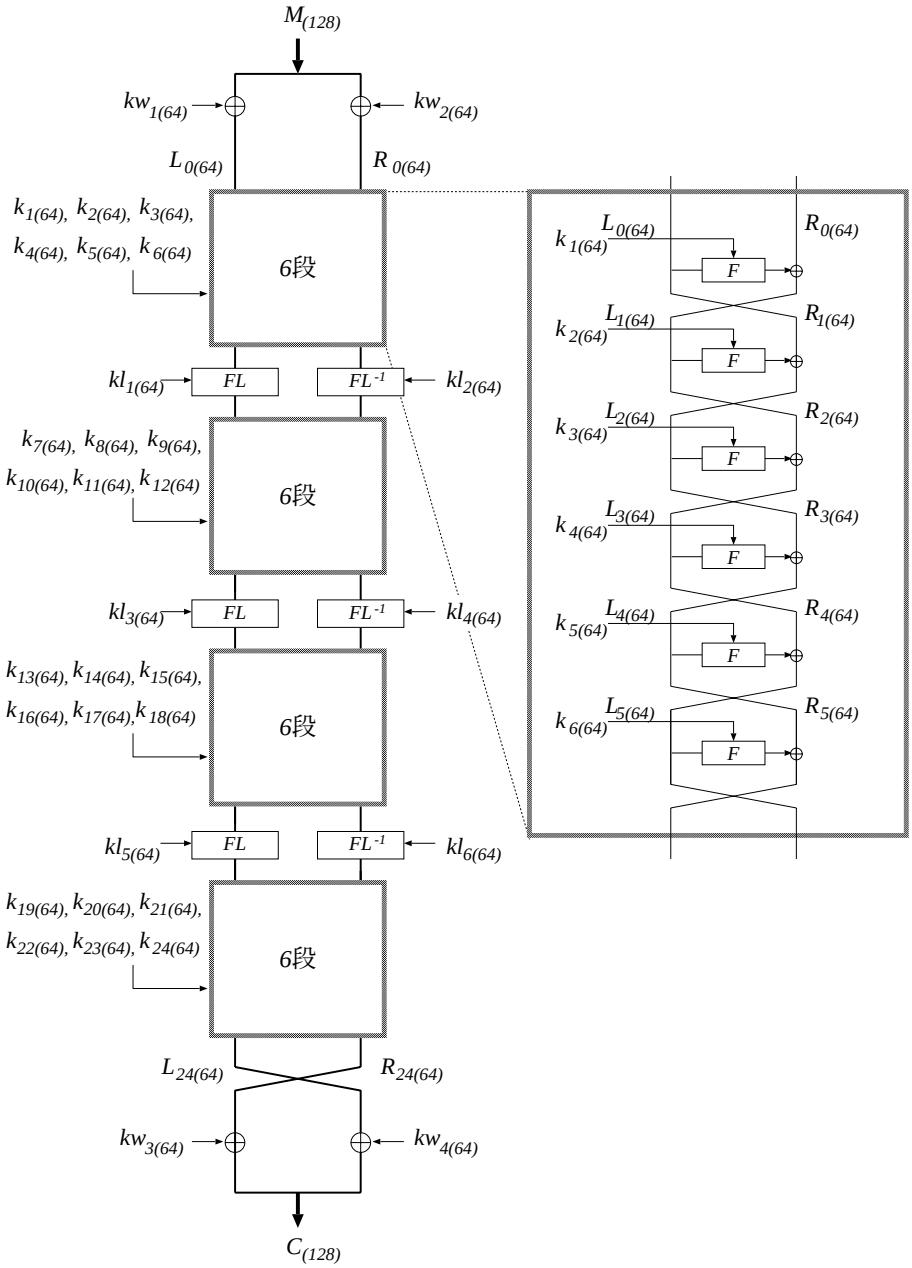

図 2: 192/256 ビット鍵 Camellia の暗号化処理

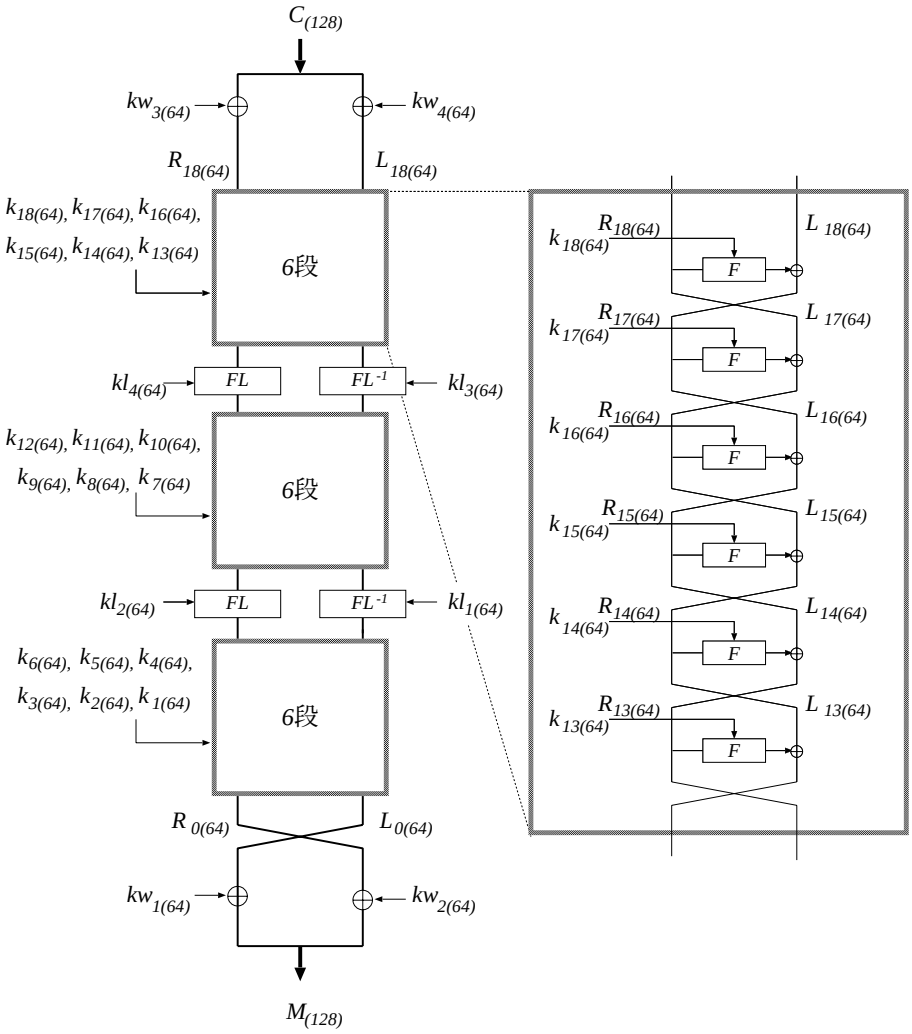

図 3: 128 ビット鍵 Camellia の復号処理

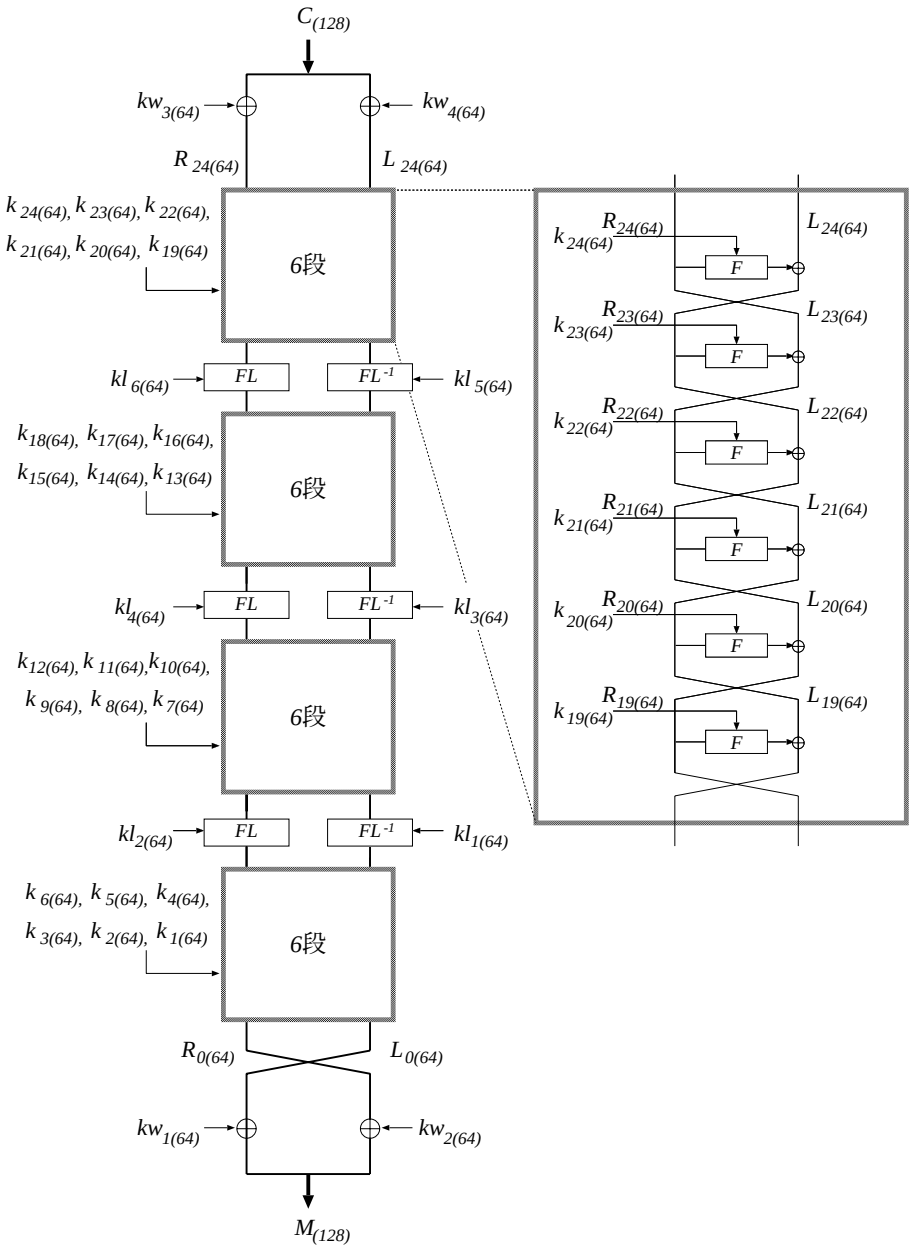

図 4: 192/256 ビット鍵 Camellia の復号処理

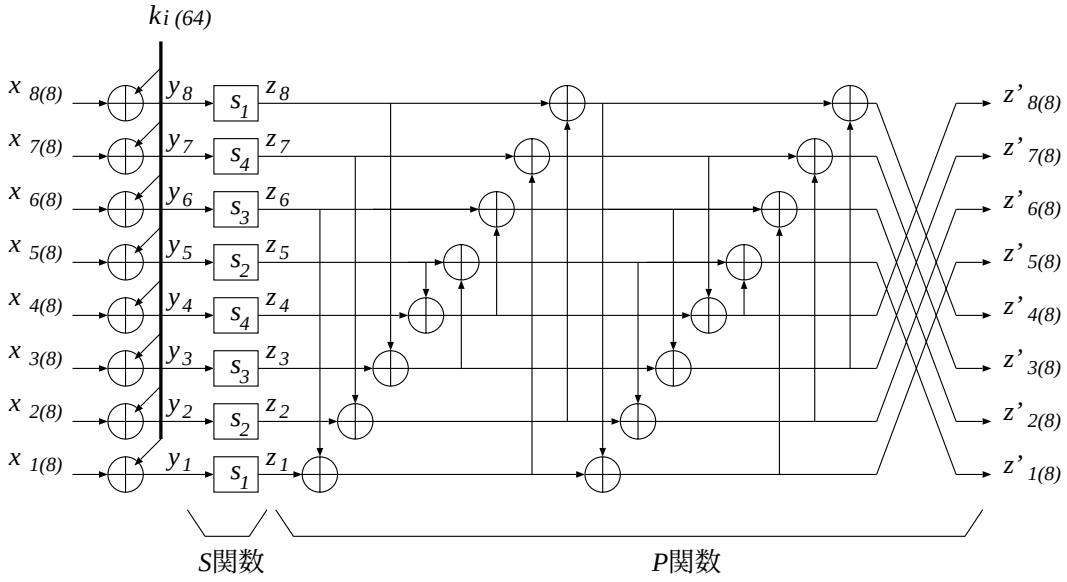

図 5: F 関数

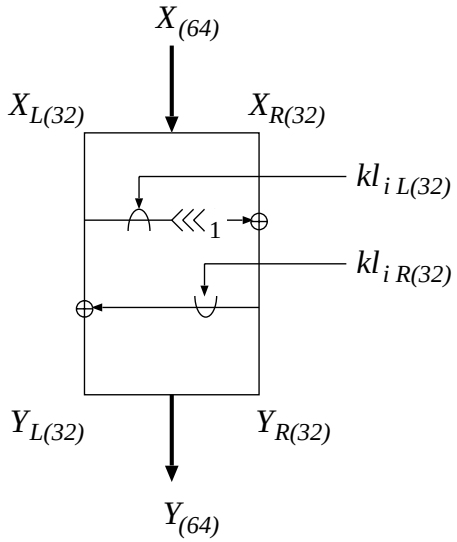

図 6: FL 関数

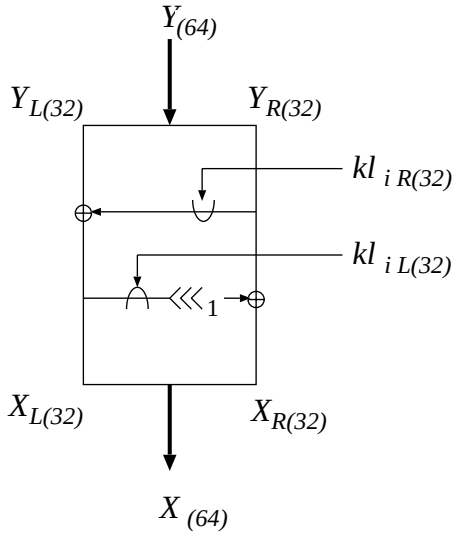

図 7: FL^{-1} 関数

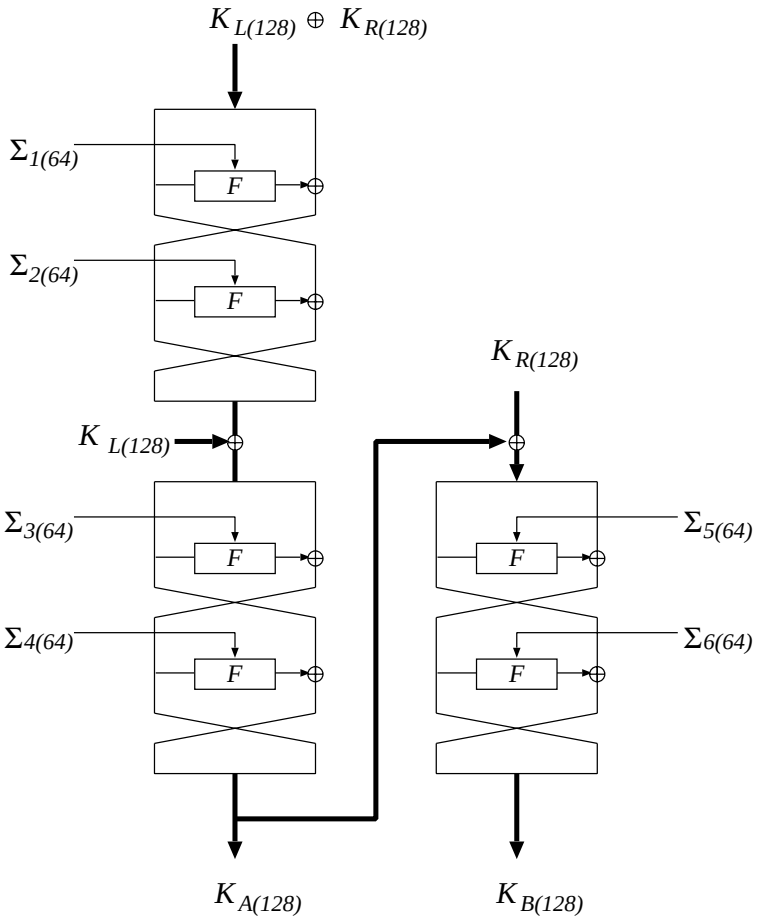
$K_{L(128)} \oplus K_{R(128)}$
$\Sigma_{1(64)}$
F
$\Sigma_{2(64)}$
F
$K_{R(128)}$
$K_{L(128)}$
$\Sigma_{3(64)}$
F
$\Sigma_{5(64)}$
F
$\Sigma_{4(64)}$
F
$\Sigma_{6(64)}$
F
$K_{A(128)}$
$K_{B(128)}$

図 8: 鍵スケジュール

B テストデータ

Camelliaのテストデータ (16進表記) は次に示す通りである。

128ビット鍵

鍵	01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10
平文	01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10
暗号文	67 67 31 38 54 96 69 73 08 57 06 56 48 ea be 43

192ビット鍵

鍵	01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10
	00 11 22 33 44 55 66 77
平文	01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10
暗号文	b4 99 34 01 b3 e9 96 f8 4e e5 ce e7 d7 9b 09 b9

256ビット鍵

鍵	01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10
	00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff
平文	01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10
暗号文	9a cc 23 7d ff 16 d7 6c 20 ef 7c 91 9e 3a 75 09

C ソフトウエアによる実装手法

この章では、Camelliaをソフトウェアで効率的に実装する方法を説明する。ほとんどの場合、プログラムは**準備部** (鍵スケジュールを含む) と**データ撹拌部** (暗号化又は復号) の2つの部分から構成される。まず準備部の最適化法を説明し、次にデータ撹拌部の最適化法を説明する。

この章では、8、32、64ビットプロセッサ向けの特別な方法も説明する。しかしながら、8ビットプロセッサ向けの最適化法は32や64ビットプロセッサにも適用可能であり、32ビットプロセッサ向けの最適化法は64ビットプロセッサにも適用可能である。他の語長も必要に応じて考慮されたい。

この章では、まずCamelliaが仕様通りに実装されていると仮定している。その後、その実装をどう最適化するかについて論じる。

この章では語長は対象プロセッサの自然な大きさを指す。例えばMMXなしのIA-32では32ビット、MMXありのIA-32やAlphaでは64ビットを指す。

C.1 準備部

C.1.1 全ての副鍵の記憶

もし、十分にメモリがあれば、一旦生成した副鍵をメモリに保存し、データ撹拌部でそれを利用する。

C.1.2 副鍵の生成順序

副鍵は必ずしも利用される順に生成しなくてもよい。例えば、128 ビット鍵の場合、まず K_L に依存する副鍵だけを生成し、次に K_A のみに依存する副鍵を生成する。そうすれば、K_A を保持しておくためのレジスタか、またはメモリを節約することができる。

C.1.3 鍵スケジュール中の排他的論理和による相殺

Camellia の鍵スケジュールは Feistel 構造に基づいている。2 段目と 3 段目の間で K_L が中間値と排他的論理和される。この構造は K_L の相殺を引き起こす。正確には、3 段目の入力は以下の式により計算できる。

$$\begin{cases} (\text{右半分}) &= F(K_{LL}, \Sigma_1) \\ (\text{左半分}) &= F(K_{LR} \oplus (\text{右半分}), \Sigma_2) \end{cases} \quad \text{128 ビット鍵}$$

$$\begin{cases} (\text{右半分}) &= K_{RR} \oplus F(K_{LL} \oplus K_{RL}, \Sigma_1) \\ (\text{左半分}) &= K_{RL} \oplus F(K_{LR} \oplus (\text{右半分}), \Sigma_2) \end{cases} \quad \text{192 ビット鍵と 256 ビット鍵}$$

上式を用いれば、128 ビット鍵の場合は 3 回、192 または 256 ビット鍵の場合は 2 回の $\mathbf{L}$ 上の排他的論理和を仕様通りの実装法に比べて削減することができる。

C.1.4 K_L, K_R, K_A, K_B の循環ビット数

副鍵生成時に、K_L, K_R, K_A, K_B について、循環シフトした値を保持しておけば元の値を保持しておく必要はない。副鍵はこの保持した値を 16 ± 1 の和だけ循環シフトすれば生成できる。

C.1.5 k_{11} と k_{12} から kl_5 と kl_6 の生成

192 と 256 ビット鍵では、(kl_5, kl_6) は $(k_{11}, k_{12}) \lll_{32}$ に等しいので (k_{11}, k_{12}) から (kl_5, kl_6) の生成にワード処理の循環シフトを利用できる。この事実を用いると、一般的な循環シフト方法に比べて、数命令削減できる。

C.1.6 動的副鍵生成

副鍵は動的に生成することが可能である。全ての副鍵は K_L, K_R, K_A, K_B のいずれかを循環シフトしたものである。従って、最初に K_L, K_R, K_A, K_B を生成し、次にそれらを循環シフトすればよい。K_L, K_R, K_A, K_B の循環シフトビット数については、第 C.1.4 章を参照のこと。

C.1.7 128 ビット鍵と 192・256 ビット鍵

もし、128 ビット鍵より長い鍵長が不要であるならば K_B を生成する必要はない。つまり、最後の 2 つの F 関数の計算を省略することができる。

C.1.8 Q 上の元をどう循環シフトするか

8 ビットプロセッサ 第 C.1.4 章に述べたように、循環シフト量の合計は 16 ± 1 の倍数の和である。16 ± 1 ビット循環シフトは 2 バイト移動の後に 1 ビットシフトすることにより実現できるので、**Q** 上の元の循環シフトは効率的に実装できる。

32 ビットプロセッサ もし IA-32 上の実装ならば、倍精度シフト命令である `shrd` や `shld` の利用を考慮されたい。

C.1.9 *F* 関数

鍵スケジュールは F 関数を含んでいるが、F 関数は主にデータ撹拌部で用いられる。第 C.2 章を参照のこと。

C.1.10 鍵依存関数

Camellia は排他的論理和と論理和と論理積の 3 つの鍵依存命令を使っている。もし可能であるならば、これらの命令に対する自己書き換えプログラムの利用も検討されたい。

C.2 データ撹拌部

C.2.1 Endian 変換

Camellia は big endian を基本に構成されている。従って、little endian などの big endian でないプロセッサでの実装は endian 変換のためのちょっとしたプログラムの追加が必要である。

もっとも安易な実装法としてはメモリからレジスタに読み込むときと、レジスタからメモリに書き出す際に endian 変換を行なう方法がある。しかしながら、FL 関数と FL^{-1} 関数のみが endian に依存している。より正確には、FL 関数や FL^{-1} 関数中の 1 ビット循環シフトのみが endian 依存である。つまり、副鍵生成法を適切に調整すれば、endian 変換を 1 ビット循環シフトとその直後に行なえばよい。1 ビット循環シフトと endian 変換を組み合わせた計算を行なうと Camellia の性能を向上できるかもしれない。この方針については、第 C.2.2 章に詳細を記した。

いくつかのプロセッサは endian 変換のための特別な命令を持っている。例えば、80486 以降の IA-32 は `bswap` を具備している。これらの命令を使用されたい。但し [C98, Appendix A] に書いてある endian 変換方法は使ってはいけない。この方法によりプログラム規模は小さくなるが、メモリ読み込みや書き込みは遅延が大きいので高速にならない。

上に記した様に、endian の問題は 32 ビットワードの 1 ビット循環シフトのみに影響する。従って、64 ビットワード全ての endian 変換をする必要はない。

以下に 32 ビットレジスタ x に対する一般的な endian 変換法を示す。下記の方法で、式中の $+$ の代わりに $\cup$ や $\oplus$ を、また、適切に mask 定数を変換すればシフトや循環シフトと論理和の計算順序を変換できることに注意されたい。

真正直な方法

$$x \leftarrow (x \ll_{24}) + ((x \cap \texttt{0xff00}) \ll_8) + ((x \gg_8) \cap \texttt{0xff00}) + (x \gg_{24})$$

この方法は並列度が高い。

循環シフトを使わない場合の命令数最小の方法

$$\begin{aligned} x &\leftarrow (x \ll_{16}) + (x \gg_{16}) \\ x &\leftarrow ((x \cap \texttt{0xff00ff}) \ll_8) + ((x \gg_8) \cap \texttt{0xff00ff}) \end{aligned}$$

循環シフトを利用する方法

$$x \leftarrow ((x \cap \texttt{0xff00ff}) \ggg_8) + ((x \lll_8) \cap \texttt{0xff00ff})$$

SSE の利用 Pentium III を含むインテルの新しいプロセッサ群は `pshufw` というデータの並び変えに非常に有効な命令が使える [I99]。`pshufw` を使うと 64 ビットデータの endian 変換はわずか 5 命令 (μops) で実現できる。

C.2.2 Little endian 解釈の 1 ビット循環シフト

第 C.2.1 章に書いたように、もし FL 関数や FL^{-1} 関数中の 1 ビット循環シフトを効率良く実装できるのであれば、平文や暗号文を読み込んだり書き出したりするときの endian 変換は必要ない。

ここで x を 1 ビット循環シフトすべき little endian でデータが格納されている 32 ビット長のレジスタとする。このとき x の 1 ビット循環シフトは以下の式で計算できる。

$$x \leftarrow ((2x) \cap \texttt{0xfefefefe}) + ((x \ggg_{15}) \cap \overline{\texttt{0xfefefefe}}) \tag{1}$$

もちろん、この方法を使うには副鍵生成や他の関数群も適切に修正する必要がある。

式 (1) 中の $+$ は $\cup$ または $\oplus$ に置き換えてもよく、また、$2x$ の計算は $\ll_1$ または $\lll_1$ または x 自身への加算に置き換えてもよい。さらに、適切に mask を変えればシフトや循環シフトと論理和の順序を入れ換えることができる。

IA-32 の `pandn` や、Alpha の `bic` の様な否定論理積 (ANDNOT) が Camellia を実装しようとしているプロセッサにあるかどうかを確認されたい。この場合は、定数 $\overline{\texttt{0xfefefefe}}$ を準備する必要はない。

C.2.3 初期及び最終排他的論理和 (whitening)

下記の式を用いれば kw_2 と kw_4 との加算は他の鍵加算に統合できる。

$$\begin{aligned} (x \oplus k) \oplus y &= (x \oplus y) \oplus k, \\ (x \oplus k) \oplus l &= x \oplus (k \oplus l), \\ (x \oplus k) \cap l &= (x \cap l) \oplus (k \cap l), \\ (x \oplus k) \lll_1 &= (x \lll_1) \oplus (k \lll_1), \\ (x \oplus k) \cup l &= (x \cup l) \oplus (k \cap \overline{l}), \end{aligned} \tag{2}$$

ここで、x, y, k, l はビット列とする。この副鍵変換を準備部で行なえば **L** 上の 2 回の排他的論理和を省略できる。

C.2.4 副鍵との排他的論理和

式 (2) を用いれば、S 関数を通過しない範囲で副鍵との排他的論理和の順序を任意の位置に変更できる。例えば、F 関数の計算を $P(S(X \oplus k))$ から $P(S(X)) \oplus k'$ と変更すれば、命令の依存性が改善されるかもしれない。

C.2.5 S 関数

s_1 は GF(2^8) 上の四則演算により定義されている。しかしながら、GF(2^8) 上の演算を計算してはいけない。代わりに事前計算し、その結果をプログラム中に埋め込むべきである。仕様の表 4 を参照のこと。

もし十分なメモリがあり、かつ 8 ビットデータの循環シフトが高コストならば、s_2, s_3, s_4 の表も s_1 に加えて事前計算し、プログラムに埋め込むことを強く推奨する。もし、十分なメモリがないのであれば、s_2, s_3, s_4 は s_1 の表と 1 回の循環シフトを使って計算されたい。(仕様の第 4.5 章参照のこと)

Java の仮想計算機 (virtual machine) のように表引き処理が重くまた十分なメモリがある環境であれば 2 つの置換表を合成した表、例えば $(s_1(y_1), s_2(y_2))$ の利用も考慮されたい。

C.2.6 P 関数

32 ビットプロセッサ $(Z_L, Z_R) = ((z_1, z_2, z_3, z_4), (z_5, z_6, z_7, z_8))$ を P 関数の入力、$(Z'_L, Z'_R) = ((z'_1, z'_2, z'_3, z'_4), (z'_5, z'_6, z'_7, z'_8))$ を P 関数の出力とする。

仕様の図 5 から P 関数は以下のように計算できることがわかる。

$$
\begin{aligned}
Z_L &\leftarrow Z_L \oplus (Z_R \lll_8) \\
Z_R &\leftarrow Z_R \oplus (Z_L \lll_{16}) \\
Z_L &\leftarrow Z_L \oplus (Z_R \ggg_8) \\
Z_R &\leftarrow Z_R \oplus (Z_L \ggg_8) \\
Z'_L &\leftarrow Z_R \\
Z'_R &\leftarrow Z_L
\end{aligned}
$$

この計算の命令依存性は高い。この計算は以下のように変更できる。

$$
\begin{array}{llll}
 & & Z_R & \leftarrow Z_R \lll_8 \\
Z_L & \leftarrow Z_L \oplus Z_R & Z_R & \leftarrow Z_R \lll_8 \\
Z_L & \leftarrow Z_L \ggg_8 & Z_R & \leftarrow Z_R \oplus Z_L \\
Z_L & \leftarrow Z_L \oplus Z_R & Z_R & \leftarrow Z_R \lll_{16} \\
Z_L & \leftarrow Z_L \lll_8 & Z_R & \leftarrow Z_R \oplus Z_L \\
Z'_L & \leftarrow Z_R & Z'_R & \leftarrow Z_L
\end{array}
$$

上記計算法の命令依存度は減少している。この方法は 1 回余分な循環シフトを必要とするように見えるが、大抵の場合、最初の計算と S 関数の計算を追加命令なしで同時に計算することができるので、実質的な命令数増加はないであろう。

8ビットプロセッサ (直交命令系) もし、排他的論理和が任意のレジスタの組合せで計算でき、十分な数のレジスタがあるならば、仕様の図 5 に書いてある方法で、P 関数を 16 回の排他的論理和で計算できる。

8ビットプロセッサ (累算器型) もし累算器型のプロセッサでの実装を考えている場合、排他的論理和数を最小化することは必ずしも良い考えとは言えない。何故ならば排他的論理和数を最小化するに当たってメモリからレジスタへの読み込みや、レジスタからメモリへの書き込みが多数必要となることがあるからである。以下の計算は、累算器型プロセッサのために最適化してある。

$$
\begin{aligned}
z'_8 &\leftarrow z_1 \oplus z_4 \oplus z_5 \oplus z_6 \oplus z_7 \\
z'_4 &\leftarrow z'_8 \oplus z_1 \oplus z_2 \oplus z_3 \\
z'_7 &\leftarrow z'_4 \oplus z_2 \oplus z_7 \oplus z_8 \\
z'_3 &\leftarrow z'_7 \oplus z_1 \oplus z_2 \oplus z_4 \\
z'_6 &\leftarrow z'_3 \oplus z_1 \oplus z_6 \oplus z_7 \\
z'_2 &\leftarrow z'_6 \oplus z_1 \oplus z_3 \oplus z_4 \\
z'_5 &\leftarrow z'_2 \oplus z_4 \oplus z_5 \oplus z_6 \\
z'_1 &\leftarrow z'_5 \oplus z_2 \oplus z_3 \oplus z_4
\end{aligned}
$$

もし、z'_i の添字調整が重い処理の場合は、以下の方法が有効である。

$$
\begin{aligned}
\sigma &\leftarrow z_1 \oplus z_2 \oplus z_3 \oplus z_4 \oplus z_5 \oplus z_6 \oplus z_7 \oplus z_8 \\
z'_1 &\leftarrow \sigma \oplus z_2 \oplus z_5 \\
z'_2 &\leftarrow \sigma \oplus z_3 \oplus z_6 \\
z'_3 &\leftarrow \sigma \oplus z_4 \oplus z_7 \\
z'_4 &\leftarrow \sigma \oplus z_1 \oplus z_8 \\
z'_5 &\leftarrow \sigma \oplus z_3 \oplus z_4 \oplus z_5 \\
z'_6 &\leftarrow \sigma \oplus z_1 \oplus z_4 \oplus z_6 \\
z'_7 &\leftarrow \sigma \oplus z_1 \oplus z_2 \oplus z_7 \\
z'_8 &\leftarrow \sigma \oplus z_2 \oplus z_3 \oplus z_8
\end{aligned}
$$

C.2.7 換字置換

この章では S と P を独立に計算するより効率良く $P \circ S$ を計算する方法について論じる。

64ビットプロセッサ もし、実装を考えているプロセッサの 1 次キャッシュが十分に大きいのであれば、文献 [RDP+96] の方法を使用されたい。この方法では、式 (3) で定義される表を用いる。

$$\begin{array}{rcl}
SP_1(y_1) & = & (s_1(y_1),\ s_1(y_1),\ s_1(y_1),\ 0,\ s_1(y_1),\ 0,\ 0,\ s_1(y_1)) \\
SP_2(y_2) & = & (\ 0,\ s_2(y_2),\ s_2(y_2),\ s_2(y_2),\ s_2(y_2),\ s_2(y_2),\ 0,\ 0) \\
SP_3(y_3) & = & (s_3(y_3),\ 0,\ s_3(y_3),\ s_3(y_3),\ 0,\ s_3(y_3),\ s_3(y_3),\ 0) \\
SP_4(y_4) & = & (s_4(y_4),\ s_4(y_4),\ 0,\ s_4(y_4),\ 0,\ 0,\ s_4(y_4),\ s_4(y_4)) \\
SP_5(y_5) & = & (\ 0,\ s_2(y_5),\ s_2(y_5),\ s_2(y_5),\ 0,\ s_2(y_5),\ s_2(y_5),\ s_2(y_5)) \\
SP_6(y_6) & = & (s_3(y_6),\ 0,\ s_3(y_6),\ s_3(y_6),\ s_3(y_6),\ 0,\ s_3(y_6),\ s_3(y_6)) \\
SP_7(y_7) & = & (s_4(y_7),\ s_4(y_7),\ 0,\ s_4(y_7),\ s_4(y_7),\ s_4(y_7),\ 0,\ s_4(y_7)) \\
SP_8(y_8) & = & (s_1(y_8),\ s_1(y_8),\ s_1(y_8),\ 0,\ s_1(y_8),\ s_1(y_8),\ s_1(y_8),\ 0)
\end{array} \tag{3}$$

次に、以下の式を計算する。

$$(z'_1, z'_2, z'_3, z'_4, z'_5, z'_6, z'_7, z'_8) \leftarrow \bigoplus_{i=1}^{8} SP_i(y_i)$$

この方法の計算量は次の通り。

表参照回数	8
排他的論理和数	7
表の大きさ (KB)	16

もし、プロセッサの 1 次キャッシュがそれなりに大きい場合は、式 (3) で定義される表のいくつかを下記の表に置き換える。

$$\begin{array}{rcl}
SP_\alpha(y) & = & (s_1(y),\ s_1(y),\ s_1(y),\ s_1(y),\ s_1(y),\ s_1(y),\ s_1(y),\ s_1(y)) \\
SP_\beta(y) & = & (s_2(y),\ s_2(y),\ s_2(y),\ s_2(y),\ s_2(y),\ s_2(y),\ s_2(y),\ s_2(y)) \\
SP_\gamma(y) & = & (s_3(y),\ s_3(y),\ s_3(y),\ s_3(y),\ s_3(y),\ s_3(y),\ s_3(y),\ s_3(y)) \\
SP_\delta(y) & = & (s_4(y),\ s_4(y),\ s_4(y),\ s_4(y),\ s_4(y),\ s_4(y),\ s_4(y),\ s_4(y))
\end{array} \tag{4}$$

次に、必要なバイト位置に 0 を埋め込む。もし式 (4) のみを利用した場合は、次の計算量を要する。

表参照回数	8
排他的論理和数	7
論理和数	8
表の大きさ (KB)	8

この方法を Alpha アーキテクチャ [C98] 上で実装しており、適切なバイト位置に 0 を埋め込むためのマスク定数を保持するためのレジスタ数が十分でないなら、`zap` または `zapnot` を利用されたい。

もし、利用するプロセッサが IA-32[I99] にある Pentium with MMX technology 以降で採用された `punpckldq` 命令や、Pentium III 以降で採用された `pshufw` 命令の様に、レジスタの半分のビットを残りの半分に効率的に複写できるのであれば、式 (3) 中の SP_1, SP_2, SP_3, SP_4 のみを準備する。そして、以下の式を計算する。

$$\begin{array}{l}
(z'_1, z'_2, z'_3, z'_4, z'_5, z'_6, z'_7, z'_8) \\
\leftarrow SP_1(y_1) \oplus SP_2(y_2) \oplus SP_3(y_3) \oplus SP_4(y_4) \oplus \nu(SP_1(y_8) \oplus SP_2(y_5) \oplus SP_3(y_6) \oplus SP_4(y_7)),
\end{array}$$

ここで、ν は最初の 4 バイトを後の 4 バイトに複写する演算とする。この方法は次の計算量を要する。

表参照回数	8
排他的論理和数	7
ν の回数	1
表の大きさ (KB)	8

32ビットプロセッサ　文献 [AU00] には Camellia 型の換字置換網の効率的な実装法が示されている。そのうちの一つの方法は、まず次に示す式 (5) を準備する。

$$\begin{aligned} SP_{1110}(y) &= (s_1(y),\ s_1(y),\ s_1(y),\ 0) \\ SP_{0222}(y) &= (0,\ s_2(y),\ s_2(y),\ s_2(y)) \\ SP_{3033}(y) &= (s_3(y),\ 0,\ s_3(y),\ s_3(y)) \\ SP_{4404}(y) &= (s_4(y),\ s_4(y),\ 0,\ s_4(y)) \end{aligned} \tag{5}$$

そして、次の式を計算する。

$$\begin{aligned} D &\leftarrow SP_{1110}(y_8) \oplus SP_{0222}(y_5) \oplus SP_{3033}(y_6) \oplus SP_{4404}(y_7) \\ U &\leftarrow SP_{1110}(y_1) \oplus SP_{0222}(y_2) \oplus SP_{3033}(y_3) \oplus SP_{4404}(y_4) \\ (z'_1, z'_2, z'_3, z'_4) &\leftarrow D \oplus U \\ (z'_5, z'_6, z'_7, z'_8) &\leftarrow (z'_1, z'_2, z'_3, z'_4) \oplus (U \ggg_8) \end{aligned}$$

この方法は、次に示す計算量を要する。

表参照回数	8
排他的論理和数	8
循環シフト数	1
表の大きさ (KB)	4

文献 [AU00] は、循環シフト処理が非常に重いプロセッサのための実装法も示している。この方法では、式 (5) で定義される表に加え次の式で定義される表も準備する。

$$\begin{aligned} SP_{1001}(y) &= (s_1(y),\ 0,\ 0,\ s_1(y)) \\ SP_{2200}(y) &= (s_2(y),\ s_2(y),\ 0,\ 0) \\ SP_{0330}(y) &= (0,\ s_3(y),\ s_3(y),\ 0) \\ SP_{0044}(y) &= (0,\ 0,\ s_4(y),\ s_4(y)) \end{aligned}$$

そして、次の式を計算する。

$$\begin{aligned} D &\leftarrow SP_{1110}(y_8) \oplus SP_{0222}(y_5) \oplus SP_{3033}(y_6) \oplus SP_{4404}(y_7) \\ (z'_1, z'_2, z'_3, z'_4) &\leftarrow D \oplus SP_{1110}(y_1) \oplus SP_{0222}(y_2) \oplus SP_{3033}(y_3) \oplus SP_{4404}(y_4) \\ (z'_5, z'_6, z'_7, z'_8) &\leftarrow D \oplus SP_{1001}(y_1) \oplus SP_{2200}(y_2) \oplus SP_{0330}(y_3) \oplus SP_{0044}(y_4) \end{aligned}$$

この方法は次の計算量を要する。

表参照回数	12
排他的論理和数	11
表の大きさ (KB)	8

C.2.8 置換表の引数生成

置換表の引数は単純にシフトと論理積を用いることにより生成できる。しかしながら、いくつかのプロセッサはこの引数を生成するための特別な命令を持っている。例えば、IA-32[I99] の `movzx` や、Alpha[C98] の `extbl` がある。

P6 で `movzx` は高速な演算であるが、下位 2 バイトに対してしか用いられない。素朴な方法として `eax`, `ebx`, `ecx`, `edx` レジスタを (L_r, R_r) の保持に用い、4 回の循環シフトを毎段実行する実装が考えられる。循環シフトのうち 2 回は、レジスタ中のバイト位置を元に戻すために必要である。しかしながら、もし循環シフトされた表を準備するのであれば毎段のバイト位置を補正するための 2 回の循環シフトを省略できる。この実装法によると、バイト位置は 4 段毎に復帰することに注意されたい。

C.3 一般的な注意事項

この章では、一般的な注意事項について述べる。これらの注意事項は、他のブロック暗号を最適化するのと同様 Camellia の最適化に際しても有効である。より詳しくは、それぞれのプロセッサに対する最適化のための取扱説明書を参照のこと。

データの整列 (align) 整列されていないデータの利用はほとんどのプロセッサにおいて、非常に重い処理である。データはワード境界に整列すべきである。

部分データの利用の回避 ほとんどのプロセッサでは、バイト処理などのワードより小さな大きさを利用するための機能を持っている。しかしながらこの機能はしばしば重い処理となっている。もし、十分なメモリを使えるのであれば、ワードの大きさが必要でないとしても、ワード利用のみにすべきである。

キャッシュの大きさへの注意 もし、プログラムやデータの大きさがキャッシュの大きさを越えると、実行速度が急激に遅くなる。ループ展開や表展開は高速化のためのよい方法であるが、キャッシュの大きさを越えないよう注意すべきである。

組み込み (intrinsic) 関数の使用 いくつかのコンパイラでは組み込み関数を利用できる。例えば、IA-32 上のマイクロソフトの Visual C++第 6 版では、「`#pragma intrinsic(_lrotl)`」と宣言し、「`_lrotl`」を利用すればコンパイラはアセンブリ言語レベルで循環シフト命令を生成する。詳細については利用しているコンパイラの取扱説明書を参照のこと。

正確な速度計測は困難 プログラムの実行速度はキャッシュ外しや、OS 割り込みなどの非常に多くの要因に依存する。さらに、何ブロック暗号化したかなどの暗号学的な要素も実行時間に影響する。

いくつかのプロセッサにはプロセッサ起動時からの経過サイクル数を得る命令がある。例えば

Pentium 以降の IA-32 は `rdtsc` 命令があり [I99]、Alpha には `rpcc` 命令がある [C98]。この様な命令を速度計測に用いることは良い考えであるが P6 や EV6 の様な非逐次実行 (out-of-order) アーキテクチャにはそのまま適用してはいけない。
もし、正確な実行時間を計測したいのであれば適切な案内書を参照されたい。例えば、Pentium 系のプロセッサについては文献 [F00] を参照のこと。

D 設計方針

日本電信電話株式会社 (以下では NTT と略す) と三菱電機株式会社 (以下では三菱電機と略す) が共同で開発した 128 ビットブロック暗号 *Camellia* を提案する。Camellia では、ブロック長が 128 ビット、鍵長が 128 または 192 または 256 ビットであり、次期米国政府標準暗号 (以下 AES; Advanced Encryption Standard) のインタフェースに準拠している。また、設計目標は以下のとおりである。

高い安全性 最近の暗号解読技術の進展には目を見張るものがある。このため、差分攻撃 [BS93] や線形攻撃 [M94] に代表される強力な暗号解読技術に対する安全性を定量的に評価することが新しい暗号を設計する上で必要不可欠なことであると考えられている。我々は、最新の暗号解読技術を利用して Camellia の安全性を評価し、2^{-128} 以上の確率を有するような差分特性や線形特性が Camallia には存在しないことを確認した。さらに、それら以外の攻撃法、例えば高階差分攻撃 [K95, JK97]、補間攻撃 [JK97, A00]、関連鍵攻撃 [B94, KSW96]、丸め差分攻撃 (truncated differential attacks) [K95, MT99]、ブーメラン攻撃 [W99]、スライド攻撃 [BW99, BW00] などに対しても安全であるように設計されている。

さまざまなプラットフォーム上での高い効率性 暗号システムはさまざまなアプリケーションで必要となるので、暗号アルゴリズムは幅広いプラットフォーム上で効率的に実装できることが望まれる。しかし、ソフトウェアとハードウェアの両面に適した 128 ビットブロック暗号アルゴリズムは数少ない。Camellia は、さまざまなプラットフォーム上での処理速度とともに、ハードウェアでのゲート数やスマートカードでの使用メモリ量などを考慮して、ハードウェアとソフトウェア両面できわめて効率的な実装が可能となるように設計されている。

Camellia は、幅広いプラットフォーム上で効率的な実装が可能である 8 ビット入出力置換表 (s-box) と論理演算から構成されている。このため、低機能 IC カードで利用されている 8 ビット CPU から、PC で広く使われている 32 ビット CPU、さらには 64 ビット CPU まで、ソフトウエアで効率的な実装が可能である。また、Camellia では、ソフトウェア実装に主眼を置いて設計されたいくつかの 128 ビットブロック暗号で広く使われている 32 ビット加算演算と乗算演算は使用しなかった。なぜなら、これらの算術演算は、Pentium II/III や Athlon のような特殊なプラットフォーム上では高速に実行されるが、それ以外のプラットフォームでは高速な処理とはいえず、また、ハードウェア実装においては、長いクリティカルパスを構成し、かつ回路規模が大きくなる原因ともなるためである。

Camellia の置換表はハードウェア規模が最小になるように設計してある。つまり、4 つの置換表は GF(2^8) 上の逆数関数のアフィン変換によって構成され、さらにその逆数関数はいくつかの GF(2^4)

上の演算によっても実現できる。これにより、置換表をより少ないゲート数で実装することが可能となっている。

鍵スケジュールは非常に簡単な構造を有し、暗号化処理の一部分を共用している。また、動的 (on-the-fly) な副鍵生成が可能であり、そのとき暗号化・復号を問わず同じ効率で副鍵が生成される。副鍵生成のためのメモリ使用量も極めて小さく、128 ビット鍵では約 32 バイトの RAM、また 192 ビット鍵と 256 ビット鍵では約 64 バイトの RAM 使用量で実装できる。

E 設計基準

E.1 *F* 関数

Camellia のラウンド関数 (以下 F 関数) の設計指針は、E2 の F 関数の設計指針 [KMA+98] を踏襲している。E2 と Camellia との主要な差異は、ラウンド関数の構造を 2 段換字置換網 (SPN; Substitution-Permutation Network) から 1 段換字置換網に変更した点である。1 段換字置換網構造を Feistel 暗号のラウンド関数に利用したとき、差分特性確率や線形特性確率の上界値による理論的評価はより複雑になるものの、差分攻撃や線形攻撃に対する実際の安全性が同じ程度としたときの暗号化処理速度が改善されると期待される。この安全性評価については第 6 章で詳細を記しているので、そちらを参照されたい。

E.2 *P* 関数

Camellia の線形変換関数 (以下 P 関数) の設計方法は、E2 の P 関数の設計方法 [KMA+98] を踏襲している。すなわち、実装効率性の観点から排他的論理和演算 (XOR) のみで構成され、また差分攻撃や線形攻撃に対する安全性の観点から分岐数 (branch number) が最良となるような線形変換関数を P 関数の候補としている。さらに、8 ビット CPU での実装のほか、32 ビット CPU [AU00] および高機能 IC カードでの高速なソフトウェア実装方法を考慮して、上記の候補の中から 1 つの P 関数を選択した。

E.3 置換表

高い安全性およびハードウェア小型化の観点から、GF(2^8) 上の逆数関数をアフィン変換した関数を利用して置換表を構成した。

GF(2^8) 上の関数における最大差分確率の最小値は 2^{-6} であることが証明されており、また最大線形確率の最小値も 2^{-6} となると予想されている。この 2^{-6} となる最良の最大差分確率と最大線形確率を達成する GF(2^8) 上の逆数関数をアフィン変換した関数が存在することから、これらの関数を置換表として採用した。このような置換表でのすべての出力ビットに対するブール多項式での次数は高いので、高階差分攻撃によって Camellia を解読することは困難である。また、GF(2^8) 上の逆数関数の入出力において実行される 2 つのアフィン関数によって置換表の GF(2^8) 上での表現が複雑になり、補間攻撃も効率的ではなくなる。さらに、4 つの異なる置換表を作ることによって、丸め差分攻撃 [MT99] に対する安全性が多少改善される。

ハードウェアの小型設計において、GF(2^8) 上の要素は部分体 GF(2^4) 上の係数の多項式としても表現することができることから、GF(2^4) 上の演算を何回か行うことによって置換表を実装することが可能である [MIYY88]。また、GF(2^8) 上の逆数関数の入出力において実行される 2 つのアフィン関数によって置換表の GF(2^4) 上での表現も複雑になる。

E.4 FL 関数と FL^{-1} 関数

FL 関数と FL^{-1} 関数は、構造の同型性を崩すために Feistel 構造の 6 段ごとに挿入される。このような関数を挿入する目的の 1 つは、現在知られていない攻撃に対する防護を図ることである。Feistel 構造の同型性が有する特徴のひとつに、暗号化処理と復号処理が副鍵の挿入順序を除いて同じ処理となっている点が挙げられる。そこで、Camellia は FL 関数と FL^{-1} 関数とを 6 段ごとに挿入するものの、上記の特性を損なわないようにしている。

FL 関数とその逆関数である FL^{-1} 関数の設計指針は、MISTY の FL 関数の設計指針を踏襲している [M97]。MISTY と Camellia との差異は、1 ビット循環シフトを加えたことである。これは、Camellia に対するバイト単位での暗号解読を難しくすることを意図すると同時に、ハードウェア規模や処理速度に対して負の影響が出ないようにするためのものである。これらの関数の設計方針は、固定された鍵に対しては (固定された) 線形変換になる一方で鍵の値によってその変換方法そのものが変わるようにすることである。すなわち、鍵が固定されている限り、これらの関数は (固定された) 線形変換となるので、平均差分確率や平均線形確率を大きくすることはない。さらに、論理積、論理和、排他的論理和 (XOR)、および循環シフトという論理演算によって構成されているので、ソフトウェアでもハードウェアでも高速な処理となっている。

E.5 鍵スケジュール

鍵スケジュールの設計指針は以下のとおりである。

1. 簡単な構成であり、さらに暗号化・復号の処理の一部分を共用できること。
2. 128 ビット鍵、192 ビット鍵、256 ビット鍵すべてについて副鍵生成回路が同様の鍵スケジュール回路で実行できること。さらに、128 ビット鍵での鍵スケジュールはその回路の一部分を利用して実現できること。
3. 副鍵生成時間は暗号化処理時間よりも短いこと。
   1 つの秘密鍵で大量のデータを暗号化する場合には鍵セットアップ時間はあまり重要ではないかもしれない。しかし、秘密鍵が頻繁に変更されるようなアプリケーションでは鍵軽快性 (key agility) が重要となる。鍵軽快性の基本的要素の一つが副鍵生成時間である。
4. 動的 (on-the-fly) 副鍵生成が可能であること。
5. 動的 (on-the-fly) 副鍵生成時に、暗号化用の副鍵生成と復号用の副鍵生成の両方が同じ効率で計算できること
   いくつかの暗号では、暗号化用の副鍵生成と復号用の副鍵生成とが異なるものがある。また、Rijndael [DR98] や Serpent [ABK98] などのように、暗号化用の副鍵は逐次的に生成できるが、復号用の副鍵は最終の暗号化用副鍵から逆順に生成しなければならないような暗号もある。

6. 等価鍵が存在しないこと。
7. 関連鍵攻撃やスライド攻撃ができないこと。

指針 1 と 2 は主にハードウェアの小型実装を目的とした項目であり、指針 3、4 および 5 は実際のアプリケーションにおける優位性を持たせるための項目である。指針 6 および 7 は安全性に関する項目である。

副鍵生成のために必要となるメモリ量は非常に小さい。128 ビット鍵での Camellia の実装では、秘密鍵 K_L 用の 16 バイト (=128 ビット) と中間鍵 K_A 用の 16 バイト (=128 ビット) との合計 32 バイトが必要なメモリ量である。同様に、192 ビット鍵および 256 ビット鍵の実装では合計 64 バイト必要となる。

F バージョン情報

Camellia は、同一の名称、かつ同一の仕様で以下に発表及び応募を行なっている。

論文発表

- 電子情報通信学会 ISEC 研究会
  青木, 市川, 神田, 松井, 盛合, 中嶋, 時田, 「128 ビットブロック暗号 Camellia」信学技報 ISEC2000-6, 2000 年 5 月.

- 国際会議 SAC 2000
  K. Aoki, T. Ichikawa, M. Kanda, M. Matsui, S. Moriai, J. Nakajima, and T. Tokita, "Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms — Design and Analysis —," In Selected Areas in Cryptography, 7th Annual International Workshop, SAC 2000, Waterloo, Ontario, Canada, August 2000, Proceedings, Lecture Notes in Computer Science 2012, pp.39-56, Springer-Verlag, 2001.

標準化機関への応募

- ISO 18033
- NESSIE
- IETF
  下記の Internet-Draft を提出.
  - J. Nakajima and S. Moriai,"A Description of the Camellia Encryption Algorithm" `<draft-nakajima-camellia-02.txt>`
  - S. Moriai,"Addition of the Camellia Encryption Algorithm to TLS" `<draft-ietf-tls-camellia-01.txt>`

G オブジェクト識別子

Camellia のオブジェクト識別子は Internet-Draft “ A Description of the Camellia Encryption Algorithm ”に記載されている。以下にその抜粋を示す。

- 鍵長 128 ビット, CBC モード
  ```
  id-camellia128-cbc OBJECT IDENTIFIER ::=
  { iso(1) member-body(2) 392 200011 61 security(1)
   algorithm(1) symmetric-encryption-algorithm(1) camellia128-cbc(2) }
  ```
- 鍵長 192 ビット, CBC モード
  ```
  id-camellia192-cbc OBJECT IDENTIFIER ::=
  { iso(1) member-body(2) 392 200011 61 security(1)
  algorithm(1) symmetric-encryption-algorithm(1) camellia192-cbc(3) }
  ```
- 鍵長 256 ビット, CBC モード
  ```
  id-camellia256-cbc OBJECT IDENTIFIER ::=
  { iso(1) member-body(2) 392 200011 61 security(1)
  algorithm(1) symmetric-encryption-algorithm(1) camellia256-cbc(4) }
  ```

H 利用実績・推奨用途など

Camelliaは共通鍵ブロック暗号が利用できるあらゆる領域に適用可能で、なかでも、暗号通信、認証に非常に適している。

Camelliaは多くのプラットフォーム上に効率良く実装可能で、PC 等で利用される 32 ビット/64 ビット CPU やローエンド/ハイエンド IC カード上でのソフトウェア実装に適するほか、ASIC, FPGA 等の小型・高速ハードウェア実装にも適している。

Camelliaの応用に関する詳細な情報は三菱電機の情報セキュリティ技術ホームページ `http://www.security.melco.co.jp/`から得ることができる。

参考文献

[A00] K. Aoki. Practical Evaluation of Security against Generalized Interpolation Attack. *IEICE Transactions Fundamentals of Electronics, Communications and Computer Sciences (Japan)*, Vol. E83-A, No. 1, pp. 33–38, 2000. (A preliminary version was presented at SAC'99).

[ABK98] R. Anderson, E. Biham, and L. Knudsen. Serpent: A Flexible Block Cipher With Maximum Assurance. In *The First AES Candidate Conference*, 1998.

[AU00] K. Aoki and H. Ueda. Optimized Software Implementations of E2. *IEICE Transactions Fundamentals of Electronics, Communications and Computer Sciences (Japan)*,

Vol. E83-A, No. 1, pp. 101–105, 2000. (The full paper is available on `http://info.isl.ntt.co.% linebreak[3]jp/e2/RelDocs/`).

[B94] E. Biham. New Types of Cryptanalytic Attacks Using Related Keys. *Journal of Cryptology*, Vol. 7, No. 4, pp. 229–246, 1994. (The extended abstract was appeared at EUROCRYPT'93).

[BS93] E. Biham and A. Shamir. *Differential Cryptanalysis of the Data Encryption Standard*. Springer-Verlag, Berlin, Heidelberg, New York, 1993.

[BW99] A. Biryukov and D. Wagner. Slide Attacks. In L. Knudsen, editor, *Fast Software Encryption — 6th International Workshop, FSE'99*, Volume 1636 of *Lecture Notes in Computer Science*, pp. 245–259, Berlin, Heidelberg, New York, 1999. Springer-Verlag.

[BW00] A. Biryukov and D. Wagner. Advanced Slide Attacks. In S. Vaudenay, editor, *Advances in Cryptology — EUROCRYPT2000*, Volume 1807 of *Lecture Notes in Computer Science*, pp. 589–606, Berlin, Heidelberg, New York, 2000. Springer-Verlag.

[C98] Compaq Computer Corporation. *Alpha Architecture Handbook (Version 4)*, 1998. (You can download the manual from Compaq's technical documentation library: `http://www.support.compaq.com/alpha-tools/documentation/current/chip-docs.html`).

[DR98] J. Daemen and V. Rijmen. *AES Proposal: Rijndael*, 1998. (`http://www.esat.kuleuven.ac.be/~rijmen/rijndael/`).

[F00] A. Fog. *How to optimize for the Pentium microprocessors*, 2000. (`http://www.agner.org/assem/`).

[I99] Intel Corporation. *Intel Architecture Software Developer's Manual (Volume 2: Instruction Set Reference)*, 1999. (You can download the manual from Intel's developer site: `http://developer.intel.com/`).

[JK97] T. Jakobsen and L. R. Knudsen. The Interpolation Attack on Block Cipher. In E. Biham, editor, *Fast Software Encryption — 4th International Workshop, FSE'97*, Volume 1267 of *Lecture Notes in Computer Science*, pp. 28–40, Berlin, Heidelberg, New York, 1997. Springer-Verlag.

[K95] L. R. Knudsen. Truncated and Higher Order Differentials. In B. Preneel, editor, *Fast Software Encryption — Second International Workshop*, Volume 1008 of *Lecture Notes in Computer Science*, pp. 196–211. Springer-Verlag, Berlin, Heidelberg, New York, 1995.

[KMA+98] M. Kanda, S. Moriai, K. Aoki, H. Ueda, M. Ohkubo, Y. Takashima, K. Ohta, and T. Matsumoto. A New 128-bit Block Cipher ***E2***. Technical Report ISEC98-12, The Institute of Electronics, Information and Communication Engineers, 1998. (in Japanese).

[KSW96] J. Kelsey, B. Schneier, and D. Wagner. Key-Schedule Cryptanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES. In N. Koblitz, editor, *Advances in Cryptology — CRYPTO'96*, Volume 1109 of *Lecture Notes in Computer Science*, pp. 237–251. Springer-Verlag, Berlin, Heidelberg, New York, 1996.

[M94] M. Matsui. Linear Cryptanalysis Method for DES Cipher. In T. Helleseth, editor, *Advances in Cryptology — EUROCRYPT'93*, Volume 765 of *Lecture Notes in Computer Science*, pp. 386–397. Springer-Verlag, Berlin, Heidelberg, New York, 1994. (A preliminary version written in Japanese was presented at SCIS93-3C).

[M97] M. Matsui. New Block Encryption Algorithm MISTY. In E. Biham, editor, *Fast Software Encryption — 4th International Workshop, FSE'97*, Volume 1267 of *Lecture Notes in Computer Science*, pp. 54–68, Berlin, Heidelberg, New York, 1997. Springer-Verlag. (A preliminary version written in Japanese was presented at ISEC96-11).

[MIYY88] M. Matsui, T. Inoue, A. Yamagishi, and H. Yoshida. A note on calculation circuits over GF(2^{2n}). Technical Report IT88-14, The Institute of Electronics, Information and Communication Engineers, 1988. (in Japanese).

[MT99] M. Matsui and T. Tokita. Cryptanalysis of a Reduced Version of the Block Cipher E2. In L. Knudsen, editor, *Fast Software Encryption — 6th International Workshop, FSE'99*, Volume 1636 of *Lecture Notes in Computer Science*, pp. 71–80, Berlin, Heidelberg, New York, 1999. Springer-Verlag. (Japanese version was presented at SCIS99.).

[RDP+96] V. Rijmen, J. Daemen, B. Preneel, A. Bosselaers, and E. De Win. The Cipher SHARK. In D. Gollmann, editor, *Fast Software Encryption — Third International Workshop*, Volume 1039 of *Lecture Notes in Computer Science*, pp. 99–111. Springer-Verlag, Berlin, Heidelberg, New York, 1996.

[W99] D. Wagner. The Boomerang Attack. In L. R. Knudsen, editor, *Fast Software Encryption — 6th International Workshop, FSE'99*, Volume 1636 of *Lecture Notes in Computer Science*, pp. 156–170, Berlin, Heidelberg, New York, 1999. Springer-Verlag.

更新履歴

- C.2.7 節　(誤)Pentium II 以降で採用された → (正)Pentium III 以降で採用された
- C.2.7 節　式 (3) 中の SP_1, SP_2, SP_3, SP_4 のみを用いて実装する場合の計算式の誤植を訂正
- D ～ H 章　追加