共通鍵ブロック暗号 SC2000

暗号技術仕様書

下山武司 †, 屋並仁史 †, 横山和弘 †, 武仲正彦 †,
伊藤孝一 †, 矢嶋純 †, 鳥居直哉 †, 田中秀磨 ‡

† 富士通研究所, ‡ 東京理科大学

2001 年 9 月 26 日

Contents

List of Figures

1 はじめに

本書は、共通鍵ブロック暗号 SC2000 の仕様に関するものである。 SC2000 のアルゴリズムは、 AES (Advanced Encryption Standard) の基本仕様と同じ 128bit によるデータ入出力で、鍵長は 128,192,256bit を使用できる。

設計方針

- 従来知られている攻撃に対する安全性を示せること。
- 様々なプラットフォームでそれぞれの性能を十分活かしたソフトウェア実装が可能であること。さらに将来の CPU で比べた場合、 AES 候補を上回る速度性能が出せる可能性があること。

安全性は暗号の最も重要な要素であり、それを何らかの形で保証できるのが望ましい。全体構造は Feistel 構造と SPN 構造の重ね合わせであり、各暗号部品には十分に安全性の検証を行った、安全性に定評がある部品のみを用いることで、全体の安全性を検証しやすい構造とし、安全性評価については、差分攻撃、線形攻撃に対する安全性はもちろん、高階差分攻撃等に対する安全性も十分考慮して設計されている。

近年の CPU の進歩はめざましいものがあるが、ブロック暗号はローエンド CPU からハイエンド CPU それぞれの持ち味を最大限活かした実装が可能であるべきだと考える。さらにできれば現在の CPU では従来の暗号に劣らない性能が実現でき、かつ最新の CPU あるいは近い将来現れるであろう CPU ではその能力を最大限活かすことにより従来暗号を上回る性能を持つ可能性があることを、安全性を損なうことなく実現することで、新暗号を提案する意味を持たせたい。これらを実現する暗号を設計する手段として本暗号では次の手段をとる。

設計手段

1. 暗号全体の構造は、 Feistel 構造と SPN 構造の重ね合わせで構成する。
2. 暗号化部においては 6 bit 入出力以下の非線形演算装置と論理演算のみを使用する。
3. SPN 構造部では Bitslice と呼ばれる高速実装法が適用可能とする。
4. Feistel 構造部では F 関数に 非均等 S-box を用いた SPN 構造を採用する。

1 および 2 によって全体として安全性評価を行ないやすい構造とし、 3 および 4 によって CPU の持つレジスタ長やキャッシュメモリーサイズに応じたソフトウエア実装のパラメータが細かく設定可能となっている。

2 記号と表記

2.1 表記

基本的に表記は以下の規則に従う。

- 入出力変数には a, b, c, d, e, f, g, h 及びそれらに数値を付加したもの ($a1$ 等) を使用する。
- 一次変数には $q, r, s, t, u, v, w, x, y, z$ 及びそれらに数値を付加したもの ($s0$ 等) を使用する。
- 変数のビット数は $a^{(4)}$ のように各変数の右肩に 10 進数で括弧付き文字で記述する。但し 32bit 変数については a のようにビット数の記述を省略する。
- 変数の各ビットの値は a_0 のように右下に 10 進数で記述する。つまり 4 bit の変数 $a^{(4)}$ の左から第 0 ビット目は $a_0^{(4)}$ と表す。
- 変数をまとめて扱う場合は (a, b, c, d) のように括弧に変数をカンマ区切りでならべて表現する。
- (a, b) のように変数をまとめて扱う場合は a を第 1 項、 b を第 2 項と表現する。
- テーブルは [ ] で表現する。例えば拡大鍵テーブルは $ek[\ ]$ と表す。
- 変数テーブルは小文字で、定数テーブルは大文字で表現する。例えば拡大鍵テーブルは $ek[\ ]$、 4bit S-Box テーブルは $S_4[\ ]$ と表す。
- 関数は出力変数 = 関数名 (入力変数) と表現する。入出力変数にはテーブルも使用できるものとする。
- 数値は Prefix がない場合 10 進数とし、 16 進数は

  `0x01234567`

  のように Prefix として `0x` をつけて表現する。

2.2 演算子と演算記号

- XOR 処理は、 2 変数のビット毎の排他的論理和で、 $a \oplus b$ と表現し、図中では $\oplus$ で表現する。
- AND 処理は、 2 変数のビット毎の論理積で、 $a \wedge b$ と表現し、図中では $\wedge$ で表現する。
- OR 処理は、 2 変数のビット毎の論理和で、 $a \vee b$ と表現し、図中では $\vee$ で表現する。
- NOT 処理は、 32bit 変数の全ビットを反転する処理で $\bar{b}$ で表現し、図中では $\bar{b}$ で表現する。
- ADD 処理は、 2 つの 32bit 変数の加算結果にたいし、 2^{32} で剰余をとる処理 $(a + b \pmod{2^{32}})$ で $a \boxplus b$ で表現し図中では $\boxplus$ で表現する。
- SUB 処理は、 2 つの 32bit 変数の減算結果にたいし、 2^{32} で剰余をとる処理 $(a - b \pmod{2^{32}})$ で $a \boxminus b$ で表現し図中では $\boxminus$ で表現する。

- MUL 処理は、 2 つの 32bit 変数の乗算結果にたいし、 2^{32} で剰余をとる処理 $(a \times b \pmod{2^{32}})$ で $a \boxtimes b$ で表現し図中では $\boxtimes$ で表現する。
- 左 1bit ローテーション処理は、 32bit 変数の 1 ビット左ローテーション処理 $((a_0, a_1, \ldots, a_{30}, a_{31}) \rightarrow (a_1, a_2, \ldots, a_{31}, a_0))$ で $a \lll_1$ で表現し図中では $\lll_1$ で表現する。
- $\lfloor x \rfloor$ は x を越えない最大の整数である。
- 繰り返し処理は C 言語に準拠した for 文で表現する。ループ変数として i, n を使用する。
- 条件分岐処理は C 言語に準拠した if 文で表現する。

2.3 Endian

Endian は Big-Endian とする。また、変数の最上位ビットを第 0 ビットとする。例えば、 $a^{(4)} = 10$ の場合、

$$(a_0^{(4)}, a_1^{(4)}, a_2^{(4)}, a_3^{(4)}) = (1, 0, 1, 0)$$

となる。

3 データ撹拌部仕様

3.1 暗号化関数

説明

暗号化関数は、 (32bit×4) の入出力である I 関数、 B 関数、 R 関数の 3 種類からなる。 128bit 鍵時の構成は、鍵を XOR する I 関数が 14 段、攪拌する関数は B 関数が 7 段、 R 関数が 12 段で合計 19 段、 192, 256bit 鍵時の構成は、鍵を XOR する I 関数が 16 段、攪拌する関数は B 関数が 8 段、 R 関数が 14 段で合計 22 段を繰り返す。使用する拡大鍵は、 128bit 鍵時は 32bit 拡大鍵が 56 個、 192, 256bit 鍵時は 64 個である。

書式

$(e, f, g, h) \ = \ encrypt(a, b, c, d, ek[\,], KeyLen)$

入力

a, b, c, d	:	32bit データ
$ek[\,]$	:	32bit 拡大鍵テーブル
$KeyLen$	:	鍵長 (128/192/256)

出力

e, f, g, h	:	32bit データ

構成

以下に暗号化関数の全体構成を示す。構成で使用する記号は次の通り。

記号	意味
(in)	入力
(out)	出力
I	I 関数
B	B 関数
$R5$	R 関数 mask=`0x55555555`
$R3$	R 関数 mask=`0x33333333`
$-$	ストレート接続 $(a, b, c, d) \rightarrow (a, b, c, d)$
$\times$	クロス接続 $(a, b, c, d) \rightarrow (c, d, a, b)$

128 bit key 時の構成:

(in)-I-B-I-$R5 \times R5$-I-B-I-$R3 \times R3$-I-B-I-$R5 \times R5$-I-B-I-$R3 \times R3$-I-B-I-$R5 \times R5$-I-B-I-$R3 \times R3$-I-B-I-(out)

192/256 bit key 時の構成:

(in)-I-B-I-$R5 \times R5$-I-B-I-$R3 \times R3$-I-B-I-$R5 \times R5$-I-B-I-$R3 \times R3$-I-B-I-$R5 \times R5$-I-B-I-$R3 \times R3$-I-B-I-$R5 \times R5$-I-B-I-(out)

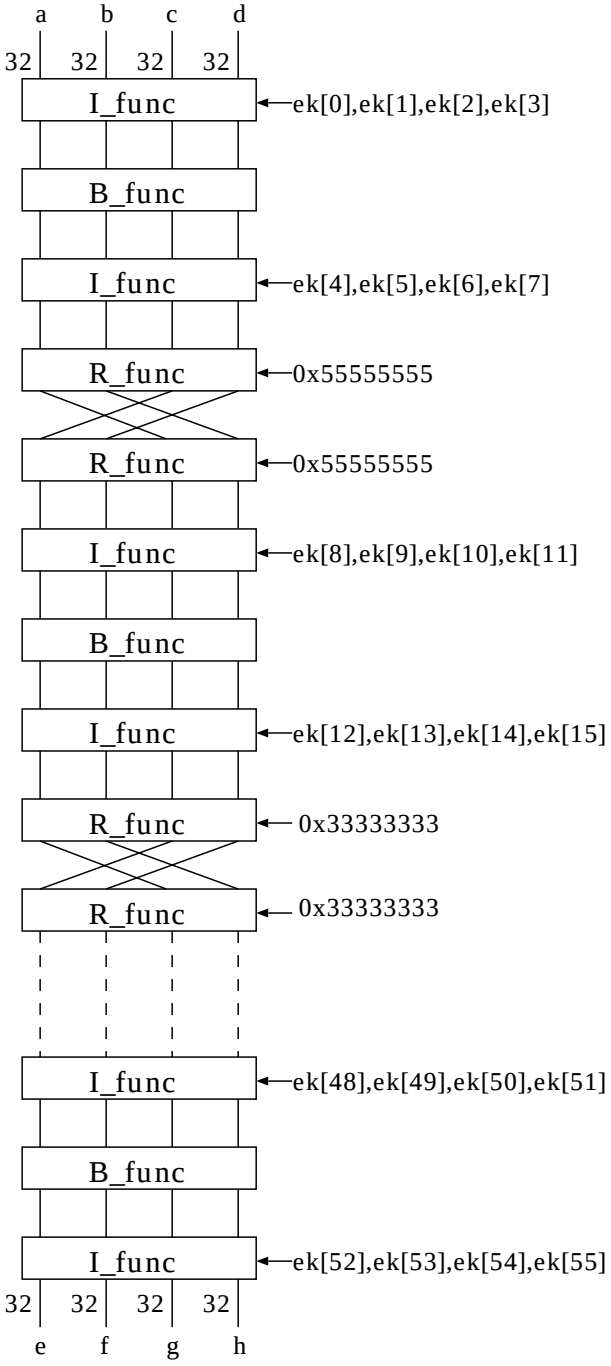

図 1: 暗号化関数 (128bit 鍵)

処理

$(s0,t0,u0,v0) = (a,b,c,d)$
$(s1,t1,u1,v1) = I\_func(s0,t0,u0,v0,\ ek[0],ek[1],ek[2],ek[3])$
$(s2,t2,u2,v2) = B\_func(s1,t1,u1,v1)$
$(s3,t3,u3,v3) = I\_func(s2,t2,u2,v2,\ ek[4],ek[5],ek[6],ek[7])$
$(s4,t4,u4,v4) = R\_func(s3,t3,u3,v3,\ 0\text{x}55555555\ )$
$(s5,t5,u5,v5) = R\_func(u4,v4,s4,t4,\ 0\text{x}55555555\ )$
$(s6,t6,u6,v6) = I\_func(s5,t5,u5,v5,\ ek[8],ek[9],ek[10],ek[11])$
$(s7,t7,u7,v7) = B\_func(s6,t6,u6,v6)$
$(s8,t8,u8,v8) = I\_func(s7,t7,u7,v7,\ ek[12],ek[13],ek[14],ek[15])$
$(s9,t9,u9,v9) = R\_func(s8,t8,u8,v8,\ 0\text{x}33333333\ )$
$(s10,t10,u10,v10) = R\_func(u9,v9,s9,t9,\ 0\text{x}33333333\ )$
$\vdots$
$(s29,t29,u29,v29) = R\_func(s28,t28,u28,v28,\ 0\text{x}33333333\ )$
$(s30,t30,u30,v30) = R\_func(u29,v29,s29,t29,\ 0\text{x}33333333\ )$
$(s31,t31,u31,v31) = I\_func(s30,t30,u30,v30,\ ek[48],ek[49],ek[50],ek[51])$
$(s32,t32,u32,v32) = B\_func(s31,t31,u31,v31)$
$(s33,t33,u33,v33) = I\_func(s32,t32,u32,v32,\ ek[52],ek[53],ek[54],ek[55])$
if $(KeyLen != 128)$ {
  $(s34,t34,u34,v34) = R\_func(s33,t33,u33,v33,\ 0\text{x}55555555\ )$
  $(s35,t35,u35,v35) = R\_func(u34,v34,s34,t34,\ 0\text{x}55555555\ )$
  $(s36,t36,u36,v36) = I\_func(s35,t35,u35,v35,\ ek[56],ek[57],ek[58],ek[59])$
  $(s37,t37,u37,v37) = B\_func(s36,t36,u36,v36)$
  $(s38,t38,u38,v38) = I\_func(s37,t37,u37,v37,\ ek[60],ek[61],ek[62],ek[63])$
  $(e,f,g,h) = (s38,t38,u38,v38)$
} else {
  $(e,f,g,h) = (s33,t33,u33,v33)$
}

3.2 復号関数

説明

復号関数は、(32bit×4) の入出力である I 関数、B^{-1} 関数、R 関数の 3 種類からなる。128bit 鍵時の構成は、鍵を XOR する I 関数が 14 段、攪拌する関数は B^{-1} 関数が 7 段、R 関数が 12 段で合計 19 段、192, 256bit 鍵時の構成は、鍵を XOR する I 関数が 16 段、攪拌する関数は B^{-1} 関数が 8 段、R 関数が 14 段で合計 22 段を繰り返す。使用する拡大鍵は、128bit 鍵時は 32bit 拡大鍵が 56 個、192, 256bit 鍵時は 32bit 拡大鍵が 64 個である。

書式

$(e,f,g,h) = decrypt(a,b,c,d,ek[\ ],KeyLen)$

入力

a,b,c,d : 32bit データ
$ek[\ ]$: 32bit 拡大鍵テーブル
$KeyLen$: 鍵長 (128/192/256)

出力

e,f,g,h : 32bit データ

構成

以下に復号関数の全体構成を示す。構成で使用する記号は次の通り。

記号	意味
(in)	入力
(out)	出力
I	I 関数
B^{-1}	B^{-1} 関数
$R5$	R 関数 mask=`0x55555555`
$R3$	R 関数 mask=`0x33333333`
$-$	ストレート接続 $(a,b,c,d) \rightarrow (a,b,c,d)$
$\times$	クロス接続 $(a,b,c,d) \rightarrow (c,d,a,b)$

128 bit key 時の構成:
(in)-I-B^{-1}-I-$R3 \times R3$-I-B^{-1}-I-$R5 \times R5$-I-B^{-1}-I-$R3 \times R3$-I-B^{-1}-I-$R5 \times R5$-I-B^{-1}-I-$R3 \times R3$-I-B^{-1}-I-$R5 \times R5$-I-B^{-1}-I-(out)

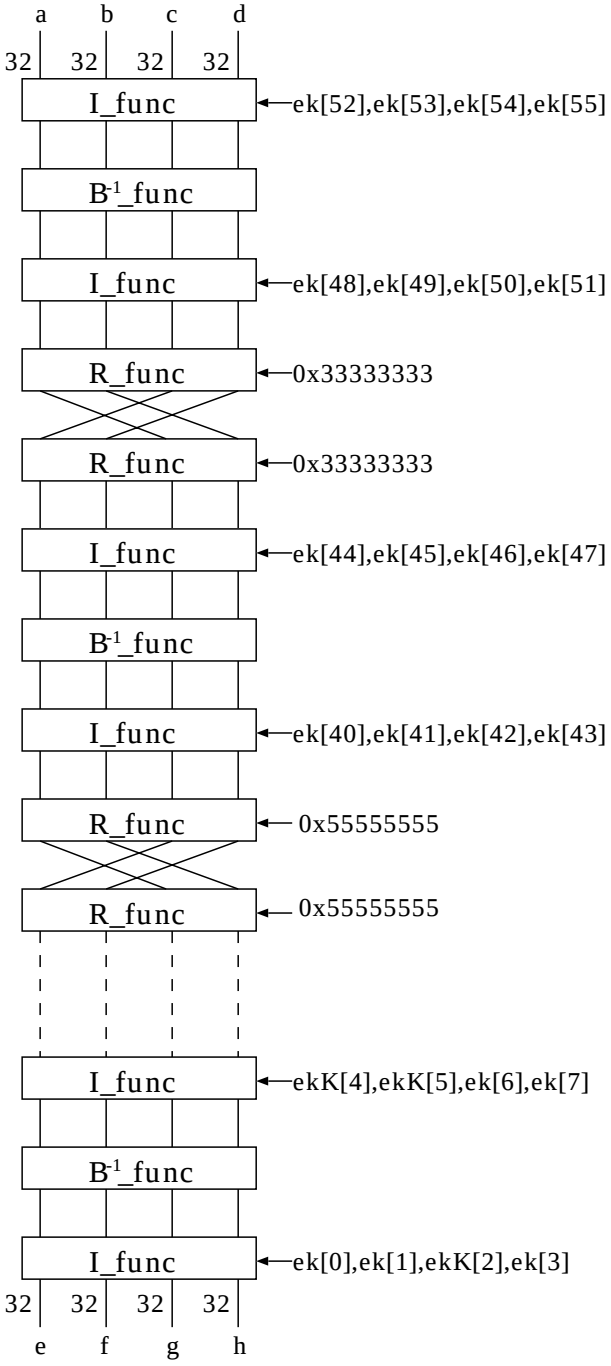

図 2: 復号関数 (128bit 鍵)

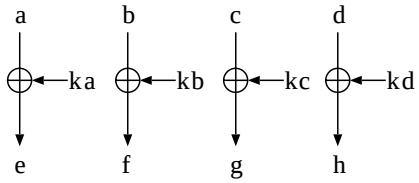

図 3: I 関数

192/256 bit key 時の構成:
(in)-I-B^{-1}-I-$R5 \times R5$-I-B^{-1}-I-$R3 \times R3$-I-B^{-1}-I-$R5 \times R5$-I-B^{-1}-I-$R3 \times R3$-I-B^{-1}-I-$R5 \times R5$-I-B^{-1}-I-$R3 \times R3$-I-B^{-1}-I-$R5 \times R5$-I-B^{-1}-I-(out)

処理

if $(KeyLen != 128)$ {
$(s38, t38, u38, v38) = (a, b, c, d)$
$(s37, t37, u37, v37) = I\_func(s38, t38, u38, v38,\ ek[60], ek[61], ek[62], ek[63])$
$(s36, t36, u36, v36) = B^{-1}\_func(s37, t37, u37, v37)$
$(s35, t35, u35, v35) = I\_func(s36, t36, u36, v36,\ ek[56], ek[57], ek[58], ek[59])$
$(s34, t34, u34, v34) = R\_func(s35, t35, u35, v35,\ 0x55555555\ )$
$(s33, t33, u33, v33) = R\_func(u34, v34, s34, t34,\ 0x55555555\ )$
} else {
$(s33, t33, u33, v33) = (a, b, c, d)$
}
$(s32, t32, u32, v32) = I\_func(s33, t33, u33, v33,\ ek[52], ek[53], ek[54], ek[55])$
$(s31, t31, u31, v31) = B^{-1}\_func(s32, t32, u32, v32)$
$(s30, t30, u30, v30) = I\_func(s31, t31, u31, v31,\ ek[48], ek[49], ek[50], ek[51])$
$(s29, t29, u29, v29) = R\_func(s30, t30, u30, v30,\ 0x33333333\ )$
$(s28, t28, u28, v28) = R\_func(u29, v29, s29, t29,\ 0x33333333\ )$
$\vdots$
$(s9, t9, u9, v9) = R\_func(s10, t10, u10, v10,\ 0x33333333\ )$
$(s8, t8, u8, v8) = R\_func(u9, v9, s9, t9,\ 0x33333333\ )$
$(s7, t7, u7, v7) = I\_func(s8, t8, u8, v8,\ ek[12], ek[13], ek[14], ek[15])$
$(s6, t6, u6, v6) = B^{-1}\_func(s7, t7, u7, v7)$
$(s5, t5, u5, v5) = I\_func(s6, t6, u6, v6,\ ek[8], ek[9], ek[10], ek[11])$
$(s4, t4, u4, v4) = R\_func(s5, t5, u5, v5,\ 0x55555555\ )$
$(s3, t3, u3, v3) = R\_func(u4, v4, s4, t4,\ 0x55555555\ )$
$(s2, t2, u2, v2) = I\_func(s3, t3, u3, v3,\ ek[4], ek[5], ek[6], ek[7])$
$(s1, t1, u1, v1) = B^{-1}\_func(s2, t2, u2, v2)$
$(s0, t0, u0, v0) = I\_func(s1, t1, u1, v1,\ ek[0], ek[1], ek[2], ek[3])$
$(e, f, g, h) = (s0, t0, u0, v0)$

3.3 I 関数

説明

32bit 変数 $\times 4$ と 32bit 拡大鍵 $\times 4$ を入力し、 32bit 変数 $\times 4$ を出力する関数。各入力データにそれぞれ拡大鍵を XOR する。拡大鍵が入力される関数は I 関数のみである。

書式

$(e, f, g, h) = I\_func(a, b, c, d, ka, kb, kc, kd)$

入力

a, b, c, d : 32bit データ
ka, kb, kc, kd : 32bit 拡大鍵データ

出力

e, f, g, h : 32bit データ

処理

$e = a \oplus ka \quad f = b \oplus kb \quad g = c \oplus kc \quad h = d \oplus kd$

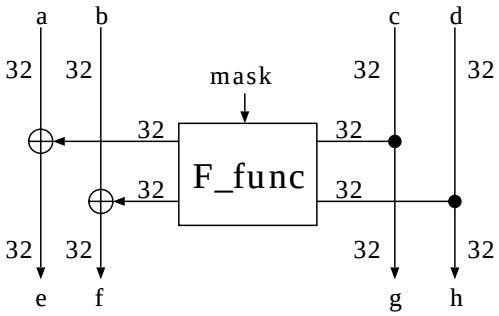

図 4: R 関数

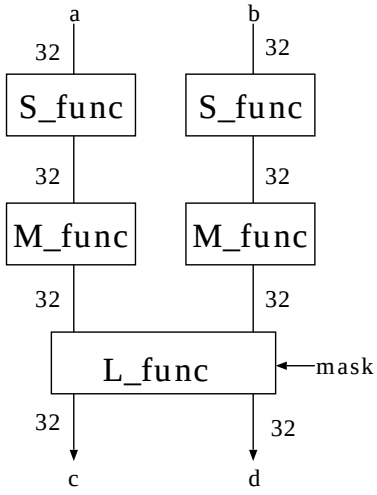

図 5: F 関数

3.4 R 関数

説明

32bit 変数 $\times 4$ 入出力の Feistel 型攪拌関数。入力データの第 3 項、第 4 項 (c,d) と定数データ $mask$ を F 関数へ入力し、 F 関数の出力 2 変数を入力データの第 1 項、第 2 項 (a,b) と XOR する。

書式

$(e,f,g,h) = R\_func(a,b,c,d,mask)$

入力

a,b,c,d : 32bit データ
$mask$: 32bit 定数データ

出力

e,f,g,h : 32bit データ

処理

$(s,t) = F\_func(c,d,mask)$
$e = a \oplus s \quad f = b \oplus t \quad g = c \quad h = d$

3.5 F 関数

説明

32bit 変数 $\times 2$ と定数を入力し 32bit 変数 $\times 2$ を出力する関数。入力の 2 変数それぞれを S 関数、 M 関数で処理し、その 2 つの出力を L 関数で処理する。

書式

$(c,d) = F\_func(a,b,mask)$

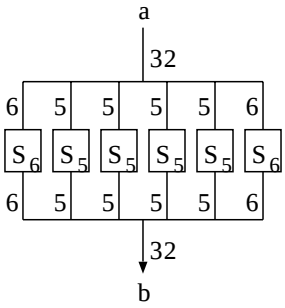

図 6: S 関数

入力

a, b : 32bit データ
$mask$: 32bit 定数データ

出力

c, d : 32bit データ

処理

$s = S\_func(a)$ $t = S\_func(b)$
$s' = M\_func(s)$ $t' = M\_func(t)$
$(c, d) = L\_func(s', t', mask)$

3.6 S 関数

説明

32bit 入出力の非線形関数。入力 32bit を (6bit, 5bit, 5bit, 5bit, 5bit, 6bit) に分割し、 6bit のものはその値で 6bit S-Box テーブル S_6 を、 5bit のものはその値で 5bit S-Box テーブル S_5 を引く。索引結果の (6bit, 5bit, 5bit, 5bit, 5bit, 6bit) を再び 32 ビットに結合して出力する。

書式

$b = S\_func(a)$

入力

a : 32bit データ

出力

b : 32bit データ

処理

$q^{(6)} = (a_0, \ldots, a_5)$ $r^{(5)} = (a_6, \ldots, a_{10})$
$s^{(5)} = (a_{11}, \ldots, a_{15})$ $t^{(5)} = (a_{16}, \ldots, a_{20})$
$u^{(5)} = (a_{21}, \ldots, a_{25})$ $v^{(6)} = (a_{26}, \ldots, a_{31})$
$q'^{(6)} = S_6[q^{(6)}]$ $r'^{(5)} = S_5[r^{(5)}]$
$s'^{(5)} = S_5[s^{(5)}]$ $t'^{(5)} = S_5[t^{(5)}]$
$u'^{(5)} = S_5[u^{(5)}]$ $v'^{(6)} = S_6[v^{(6)}]$
$(b_0, \ldots, b_5) = q'^{(6)}$ $(b_6, \ldots, b_{10}) = r'^{(5)}$
$(b_{11}, \ldots, b_{15}) = s'^{(5)}$ $(b_{16}, \ldots, b_{20}) = t'^{(5)}$
$(b_{21}, \ldots, b_{25}) = u'^{(5)}$ $(b_{26}, \ldots, b_{31}) = v'^{(6)}$

3.7 M 関数

説明

32bit 入出力の線形関数。入力を 1bit×32 のベクトルと見なし、 32×32bit のテーブル M (すなわち 32×32×1bit の行列) と乗算を行う。

書式

$b = M\_func(a)$

入力

a : 32bit データ

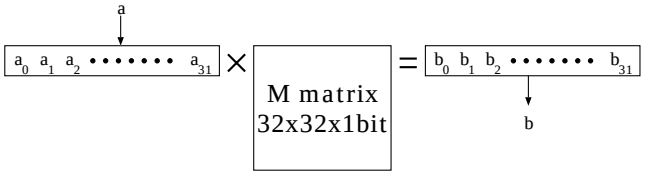

図 7: M 関数

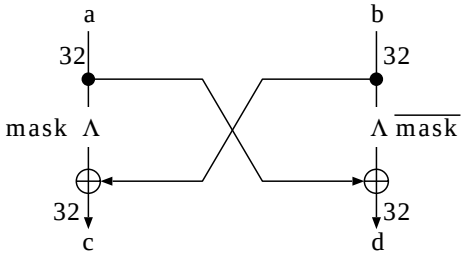

図 8: L 関数

出力

b : 32bit データ

処理

$for\ (b=0, i=0; i<32; i++)\ \{$
$\quad if\ (a_i == 1)\ b = b \oplus M[i]$
$\}$

3.8 L 関数

説明

32bit 変数 ×2 と定数を入力し 32bit 変数 ×2 出力する関数。2 つの入力値 (a,b) に対して a と定数の AND 値に b を XOR したものと、b と定数のビット反転した値を AND した値に a を XOR したものを出力する。

書式

$(c,d) = L\_func(a,b,mask)$

入力

a,b : 32bit データ
$mask$: 32bit 定数データ

出力

c,d : 32bit データ

処理

$s = a \wedge mask \quad t = b \wedge \overline{mask}$
$c = s \oplus b \qquad d = t \oplus a$

3.9 B/B^{-1} 関数

説明

32bit×4 の入力データを、T 関数を用いて 4bit×32 に変換し、それぞれの値を 4bit S-Box テーブル S_4 を用いて変換する。そして変換された 4bit×32 に対し、T^{-1} 関数を用いて 32bit×4 に変換し、出力する。B^{-1} 関数では、4bit S-Box テーブル S_4 の逆変換 $S_4 i$ を用いる。

書式

$(e,f,g,h) = B\_func(a,b,c,d)$
$(e,f,g,h) = B^{-1}\_func(a,b,c,d)$

入力

a,b,c,d : 32bit データ

出力

e,f,g,h : 32bit データ

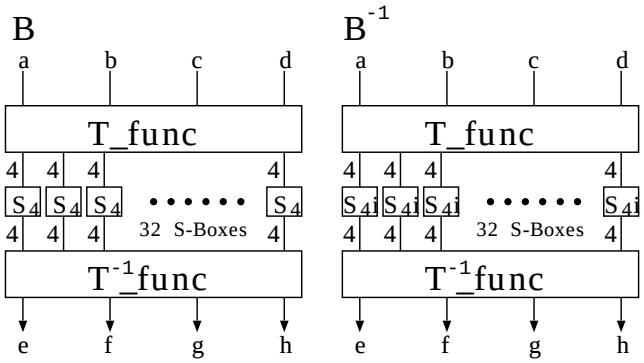

図 9: B, B^{-1} 関数

処理

B_func:
$(s0^{(4)}, s1^{(4)}, ..., s31^{(4)}) = T\_func(a, b, c, d)$
$t0^{(4)} = S_4[s0^{(4)}] \qquad t1^{(4)} = S_4[s1^{(4)}]$
$\vdots$
$t30^{(4)} = S_4[s30^{(4)}] \qquad t31^{(4)} = S_4[s31^{(4)}]$
$(e, f, g, h) = T^{-1}\_func(t0^{(4)}, t1^{(4)}, ..., t31^{(4)})$

B^{-1}_func:
$(s0^{(4)}, s1^{(4)}, ..., s31^{(4)}) = T\_func(a, b, c, d)$
$t0^{(4)} = S_4i[s0^{(4)}] \qquad t1^{(4)} = S_4i[s1^{(4)}]$
$\vdots$
$t30^{(4)} = S_4i[s30^{(4)}] \qquad t31^{(4)} = S_4i[s31^{(4)}]$
$(e, f, g, h) = T^{-1}\_func(t0^{(4)}, t1^{(4)}, ..., t31^{(4)})$

3.10 T/T^{-1} 関数

説明

T 関数は 32bit データ 4 つを 32×4 の 1bit 行列と見なし、それを転置して 4×32 の行列に変換し、 4bit データ 32 個とする。 T^{-1} 関数は T の逆関数で 4bit データ 32 個を 4×32 の行列と見なし、それを転置して 32×4 の行列に変換し、32bit データ 4 個とする。

書式

$(s0^{(4)}, s1^{(4)}, \ldots s31^{(4)}) = T\_func(a, b, c, d)$
$(e, f, g, h) = T^{-1}\_func(t0^{(4)}, t1^{(4)}, ...t31^{(4)})$

入力

a, b, c, d : 32bit データ
$t0^{(4)}, t1^{(4)}, \ldots t31^{(4)}$: 4bit データ

出力

$s0^{(4)}, s1^{(4)}, \ldots s31^{(4)}$: 4bit データ
e, f, g, h : 32bit データ

処理

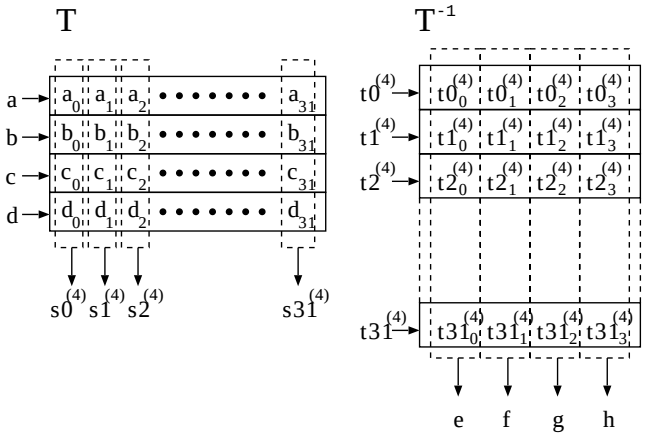

図 10: T, T^{-1} 関数

T_func:
$s0^{(4)} = (s0_0^{(4)}, s0_1^{(4)}, s0_2^{(4)}, s0_3^{(4)}) = (a_0, b_0, c_0, d_0)$
$s1^{(4)} = (s1_0^{(4)}, s1_1^{(4)}, s1_2^{(4)}, s1_3^{(4)}) = (a_1, b_1, c_1, d_1)$
$\vdots$
$s31^{(4)} = (s31_0^{(4)}, s31_1^{(4)}, s31_2^{(4)}, s31_3^{(4)}) = (a_{31}, b_{31}, c_{31}, d_{31})$

T^{-1}_func:
$e = (e_0, e_1, ..., e_{31}) = (t0_0^{(4)}, t1_0^{(4)}, ..., t31_0^{(4)})$
$f = (f_0, f_1, ..., f_{31}) = (t0_1^{(4)}, t1_1^{(4)}, ..., t31_1^{(4)})$
$g = (g_0, g_1, ..., g_{31}) = (t0_2^{(4)}, t1_2^{(4)}, ..., t31_2^{(4)})$
$h = (h_0, h_1, ..., h_{31}) = (t0_3^{(4)}, t1_3^{(4)}, ..., t31_3^{(4)})$

4 鍵スケジュール部仕様

4.1 鍵スケジュール部の全体構成

説明

ユーザ鍵から 32bit 拡大鍵 56 個 (鍵長 128bit 時)、 64 個 (鍵長 192bit,256bit 時) を生成する、中間鍵生成関数と拡大鍵生成関数からなる関数。鍵長 128bit 時は、 $uk[4] = uk[0], uk[5] = uk[1], uk[6] = uk[2], uk[7] = uk[3]$ とユーザ鍵を 8 個に拡張、同様に鍵長 192bit 時は、 $uk[6] = uk[0], uk[7] = uk[1]$ と拡張して、中間鍵生成関数を実行する。中間鍵生成関数で中間鍵 $imkey[\ ]$ を生成し、つぎに拡大鍵生成関数を実行することで 32bit の拡大鍵 $ek[\ ]$ を生成する。生成する拡大鍵は、鍵長 128bit 時 56 個、 192bit,256bit 時 64 個である。

書式

$(ek[\,]) = make\_keys(uk[\,], KeyLen)$

入力

$uk[\ ]$: 32bit ユーザ鍵テーブル (鍵長 128bit 時 4 個、 192bit 時 6 個、 256bit 時 8 個)
$KeyLen$: 鍵長 (128/192/256)

出力

$ek[\ ]$: 32bit 拡大鍵 (鍵長 128bit 時 56 個、 192bit,256bit 時 64 個)

処理

if ($KeyLen == 128$) {
  uk[4] = uk[0] uk[5] = uk[1] uk[6] = uk[2] uk[7] = uk[3]
} else if ($KeyLen == 192$) {
  uk[6] = uk[0] uk[7] = uk[1] }
$(a[\ ], b[\ ], c[\ ], d[\ ]) = make\_imkeys(uk[\ ])$
$(ek[\ ]) = make\_ekeys(a[\ ], b[\ ], c[\ ], d[\ ], KeyLen)$

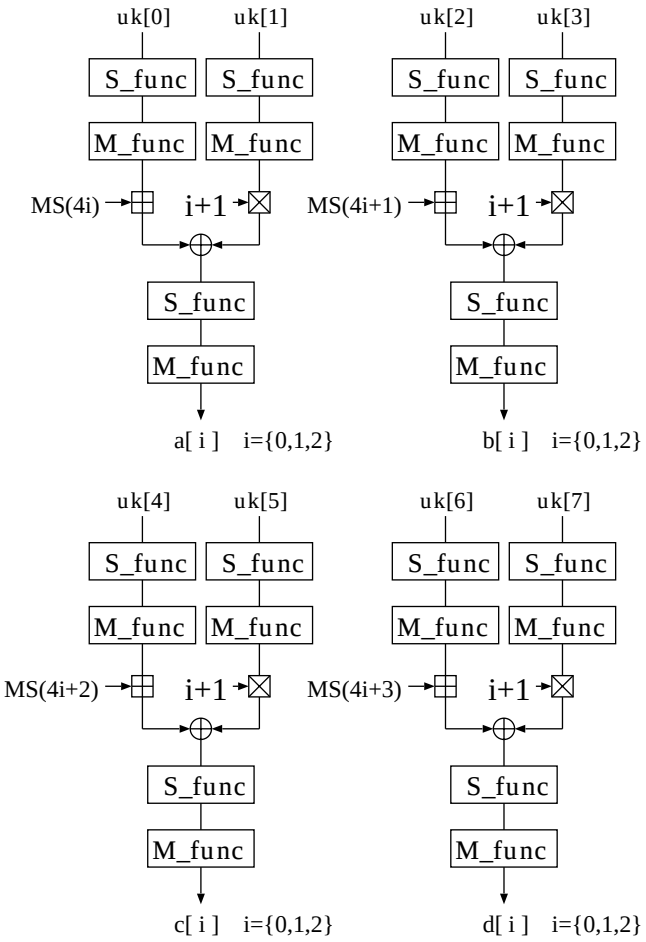

図 11: 中間鍵生成関数

4.2 中間鍵生成関数

説明

ユーザ鍵 (32bit×8 個) から中間鍵 (32bit×12 個) を作成する。

書式

$(a[\,],b[\,],c[\,],d[\,]) = make\_imkeys(uk[\,])$

入力

$uk[\,]$: 32bit ユーザ鍵テーブル (8 個)

出力

$a[\,],b[\,],c[\,],d[\,]$: 中間鍵テーブル(各3個)

処理

for $(i = 0; i < 3; i++)$ {

$$a[i] = M\_func(S\_func((M\_func(S\_func(4i)) \boxplus M\_func(S\_func(uk[0]))) \oplus (M\_func(S\_func(uk[1])) \boxtimes (i+1))))$$

$$b[i] = M\_func(S\_func((M\_func(S\_func(4i+1)) \boxplus M\_func(S\_func(uk[2]))) \oplus (M\_func(S\_func(uk[3])) \boxtimes (i+1))))$$

$$c[i] = M\_func(S\_func((M\_func(S\_func(4i+2)) \boxplus M\_func(S\_func(uk[4]))) \oplus (M\_func(S\_func(uk[5])) \boxtimes (i+1))))$$

$$d[i] = M\_func(S\_func((M\_func(S\_func(4i+3)) \boxplus M\_func(S\_func(uk[6]))) \oplus (M\_func(S\_func(uk[7])) \boxtimes (i+1))))$$

}

4.3 拡大鍵生成関数

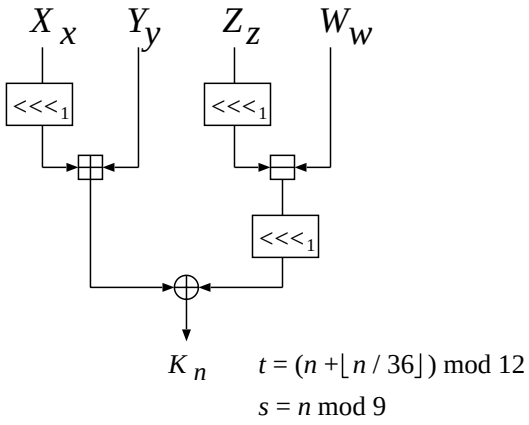

図 12: 拡大鍵生成関数

説明

中間鍵 (32bit×12 個) から 32 bit 拡大鍵 56 個 (鍵長 128 bit 時)、 64 個 (鍵長 192bit,256bit 時) を生成する。

書式

$(ek[\ ]) = make\_ekeys(a[\ ], b[\ ], c[\ ], d[\ ], KeyLen)$

入力

$a[\ ], b[\ ], c[\ ], d[\ ]$: 中間鍵テーブル (各 3 個)

$KeyLen$: 鍵長 (128/192/256)

出力

$ek[\ ]$: 32bit 拡大鍵
(鍵長 128bit 時 56 個、 192bit,256bit 時 64 個)

処理

if $(KeyLen == 128)\ num\_ekey = 56$
else $num\_ekey = 64$
for $(n = 0; n < num\_ekey; n++)$ {

$s = n \pmod 9$
$t = (n + \lfloor n/36 \rfloor) \pmod{12}$
$X = Order[t][0] \quad x = Index[s][0]$
$Y = Order[t][1] \quad y = Index[s][1]$
$Z = Order[t][2] \quad z = Index[s][2]$
$W = Order[t][3] \quad w = Index[s][3]$
$ek[n] = ((X[x] \lll_1) \boxplus Y[y]) \oplus (((Z[z] \lll_1) \boxminus W[w]) \lll_1)$

}

5 テーブル

本暗号アルゴリズムで使用するテーブルを示す。

5.1 データ撹拌部

```
 /* S-box */
 S6[64] =
 {47,59,25,42,15,23,28,39,26,38,36,19,60,24,29,56,
  37,63,20,61,55, 2,30,44, 9,10, 6,22,53,48,51,11,
  62,52,35,18,14,46, 0,54,17,40,27, 4,31, 8, 5,12,
   3,16,41,34,33, 7,45,49,50,58, 1,21,43,57,32,13};
 S5[32] =
 {20,26, 7,31,19,12,10,15,22,30,13,14, 4,24, 9,18,
  27,11, 1,21, 6,16, 2,28,23, 5, 8, 3, 0,17,29,25};
 S4[16] =
 { 2, 5,10,12, 7,15,1,11,13, 6, 0, 9, 4, 8, 3,14};
 S4i[16] =
 {10, 6, 0,14,12, 1, 9,4,13,11, 2, 7, 3, 8,15, 5};

/* M-Table */
 M[32] =
 {0xd0c19225,0xa5a2240a,0x1b84d250,0xb728a4a1,
  0x6a704902,0x85dddbe6,0x766ff4a4,0xecdfe128,
  0xafd13e94,0xdf837d09,0xbb27fa52,0x695059ac,
  0x52a1bb58,0xcc322f1d,0x1844565b,0xb4a8acf6,
  0x34235438,0x6847a851,0xe48c0cbb,0xcd181136,
  0x9a112a0c,0x43ec6d0e,0x87d8d27d,0x487dc995,
  0x90fb9b4b,0xa1f63697,0xfc513ed9,0x78a37d93,
  0x8d16c5df,0x9e0c8bbe,0x3c381f7c,0xe9fb0779};
```

5.2 鍵スケジュール部

Order

t	X	Y	Z	W
0	a	b	c	d
1	b	a	d	c
2	c	d	a	b
3	d	c	b	a
4	a	c	d	b
5	b	d	c	a
6	c	a	b	d
7	d	b	a	c
8	a	d	b	c
9	b	c	a	d
10	c	b	d	a
11	d	a	c	b

Index

s	x	y	z	w
0	0	0	0	0
1	1	1	1	1
2	2	2	2	2
3	0	1	0	1
4	1	2	1	2
5	2	0	2	0
6	0	2	0	2
7	1	0	1	0
8	2	1	2	1

6 ソフトウェア実装高速化手法

本暗号アルゴリズムを高速処理するための手法について述べる。実装するプラットフォームに応じて高速化手法を取捨選択することで実装環境に最適な処理が可能となる。

6.1 データ撹拌部

6.1.1 テーブルの結合

3.6節の S 関数の説明では 5bit と 6bit の S-Box テーブルを (6,5,5,5,5,6) という形で索引しているが、隣接する S-Box を 1 つの S-Box と見なして (6,10,10,6) や (11,10,11) 等の形でテーブル索引を行うことで、索引回数を削減でき、高速化が可能となる。

6.1.2 関数の結合

F 関数内では S 関数、 M 関数、 L 関数の順で処理を行っているが、 S 関数と M 関数はどちらもテーブル索引を行う関数なので、この 2 つの関数は結合処理するが可能である。この場合、テーブルを結合して S_5_M(5bit 入力 32bit 出力) や S_6_M(6bit 入力 32bit 出力) を作成することで、テーブルの索引回数を削減でき、高速化が可能となる。さらに、 L 関数とも結合して $S_5\_M\_L$(5bit 入力 64bit 出力) や $S_6\_M\_L$(6bit 入力 64bit 出力) として高速化が可能である。この手法は 6.1.1節のテーブルの結合とも併用可能でその場合 S_{10}_Mや$S_{11}_M_L$ といったテーブルとなる。

6.1.3 64bit 処理

I 関数と R 関数は 32bit 入出力 4 つを 2 つずつまとめて 64bit 処理を行うことで高速化可能である。 I 関数では鍵テーブルの読み込みの数を削減できる。また R 関数では、 6.1.2節で述べた S 関数、 M 関数、 L 関数の結合で 64bit に適したものとなる。

6.2 ビットスライス処理

B/B^{-1} 関数は、 S_4 テーブルを論理式で表現することによりビットスライス処理が可能である。ビットスライス処理を行えば、 32 個の S-Box 索引をまとめて処理することが可能となり高速化できる。例えば、 B/B^{-1} 関数は次の様に論理式で記述できる。

なお B 関数、 B^{-1} 関数の論理式表現はプロセッサの持つレジスタ数、パイプライン数等によってそれぞれ適した表現を求める必要がある。現在使われている汎用のプロセッサを考慮して与えられた次の二種類の目標から探索された論理表現で、本論文の執筆時点で得られている最良性能値を表 2 に記す。

(1) RISC 系プロセッサのアセンブラ命令でソースオペランドとディスティネーションオペランドが分離している 3 オペランド命令セットを前提に、レジスタ数は制限を設けずに演算数最少のものを見つける探索
(2) x86 系 (Intel 系) プロセッサの 2 オペランドアセンブラ命令セットを前提にレジスタ数が 6 以内で演算数最少のものを見つける探索

6.3 鍵スケジュール部

6.3.1 定数の事前計算

中間鍵生成関数には数値 $(4i, 4i+1, 4i+2, 4i+3)$ $\{i = 0, 1, 2\}$ を S_func, M_func 処理する部分が存在する。この部分は鍵によらない値であるので $M\_func(S\_func(4i))$ 等 12 個の値を事前計算することで、計算量を削減できる。

6.3.2 値の再利用

中間鍵生成関数には数値 $uk[0] \sim uk[7]$ を S_func, M_func 処理する部分が存在する。この部分は値 i に拠らないため、 $M\_func(S\_func(uk[0])) \sim M\_func(S\_func(uk[7]))$ の値を保存しておけば、値 i が更新されても再計算の必要はなく、計算量の削減ができる。本手法と 6.3.1節で述べた定数の事前計算を行えば、 S_func, M_func の計算回数 48 回が 20 回に削減できる。

6.3.3 乗算処理の代替

中間鍵生成関数には乗算処理があるが、乗数が 1, 2, 3 のみであることから、この乗算処理を加算処理で代替することで高速化が可能となる場合がある。

表 1: B 関数,B^{-1} 関数 のビットスライス実装

$(e,f,g,h) = B\_func(a,b,c,d)$

$t1 = \overline{b}$
$t2 = a \vee d$
$t3 = c \oplus t2$
$t4 = t1 \wedge t3$
$e = d \oplus t4$
$t6 = t2 \oplus e$
$t7 = a \oplus b$
$t8 = t6 \vee t7$
$f = c \oplus t8$
$t10 = c \wedge t3$
$t11 = t1 \oplus t10$
$t12 = c \oplus t4$
$t13 = t6 \wedge t12$
$h = t7 \oplus t13$
$t15 = c \vee h$
$g = t11 \oplus t15$

$(e,f,g,h) = B^{-1}\_func(a,b,c,d)$

$t1 = c \oplus d$
$t2 = t1 \vee b$
$t3 = t2 \oplus c$
$t4 = t3 \wedge a$
$t5 = \overline{c}$
$t6 = t5 \vee b$
$t7 = t6 \oplus d$
$e = t7 \oplus t4$
$t8 = a \oplus t6$
$t9 = t8 \vee d$
$g = t9 \oplus b$
$t10 = c \oplus g$
$t11 = t10 \vee t4$
$t12 = t11 \wedge t9$
$f = t12 \oplus e$
$t13 = t6 \oplus g$
$t14 = t13 \wedge t1$
$h = t14 \oplus a$

表 2: B 関数,B^{-1} 関数の論理式表現の性能値

探索	関数	並列度	レジスタ数	総演算数	サイクル数
(1)	B	2	8	16	8
(1)	B^{-1}	2	9	16	11
(2)	B	2	5	19	12
(2)	B^{-1}	2	6	23	13

Appendix A テストベクトル

本暗号アルゴリズム 128bit 鍵処理時における中間鍵、拡大鍵、および、中間値テストベクトルを以下に示す。値は、各関数処理後の値である。

```
KEYSIZE=128
KEY=0x00000000000000000000000000000000

InterMediate Key:
A[0]=0x1E13B607, A[1]=0x8037CF49, A[2]=0x7A1DC8C8
B[0]=0xFFECA80A, B[1]=0xF2062F0C, B[2]=0x224A06EC
C[0]=0x968A7468, C[1]=0x064C0837, C[2]=0xDF864B05
D[0]=0x79FA438A, D[1]=0x937B7E58, D[2]=0xE4ABD855

Extended Key:
EXKEY[0]=0x5A215E97, EXKEY[1]=0x2511C596
EXKEY[2]=0x005B7B29, EXKEY[3]=0x05038ED3
EXKEY[4]=0xD6AC0212, EXKEY[5]=0xFF7F916B
EXKEY[6]=0x91685E19, EXKEY[7]=0xF529F0ED
EXKEY[8]=0xFB2704AA, EXKEY[9]=0x12399574
EXKEY[10]=0xB3E065AB, EXKEY[11]=0x7AF0674C
EXKEY[12]=0x1D1F4FE9, EXKEY[13]=0xD0CB45B9
EXKEY[14]=0xD19B0A98, EXKEY[15]=0xD80DDC82
EXKEY[16]=0xD8EEBBB5, EXKEY[17]=0xA5A601B4
EXKEY[18]=0x409687CF, EXKEY[19]=0xECBA0704
EXKEY[20]=0x12FCEC43, EXKEY[21]=0x57728321
EXKEY[22]=0x77507089, EXKEY[23]=0x9954BAB1
EXKEY[24]=0xCEA35202, EXKEY[25]=0x22F904B3
EXKEY[26]=0x56E2D16B, EXKEY[27]=0x49F5CF61
EXKEY[28]=0x6F5A3D80, EXKEY[29]=0xA0E27CAB
EXKEY[30]=0x75F71B60, EXKEY[31]=0x0893A481
EXKEY[32]=0x3226E7FB, EXKEY[33]=0x71A8BC68
EXKEY[34]=0x1D42352C, EXKEY[35]=0xD383B20B
EXKEY[36]=0xA7392344, EXKEY[37]=0xBCC151C8
EXKEY[38]=0x3C317191, EXKEY[39]=0x41AFC455
EXKEY[40]=0xEC4CB923, EXKEY[41]=0x4813D88F
EXKEY[42]=0xAF7CCC12, EXKEY[43]=0xE16A317F
EXKEY[44]=0x8B60307F, EXKEY[45]=0x86C032C0
EXKEY[46]=0x920D093E, EXKEY[47]=0xA244E311
EXKEY[48]=0x5B41E2E5, EXKEY[49]=0x4D08C78C
EXKEY[50]=0x12E28AB1, EXKEY[51]=0xB8F8B742
EXKEY[52]=0x830E156C, EXKEY[53]=0x5D757A55
EXKEY[54]=0xB8D8C053, EXKEY[55]=0x286BB72E

PT=0x00000000000000000000000000000000

Encrypt
I :0x5A215E97 0x2511C596 0x005B7B29 0x05038ED3
B :0x5F6BAEBB 0x7F238044 0xA5CDE028 0x7F30CB41
I :0x89C7ACA9 0x805C112F 0x34A5BE31 0x8A193BAC
R5:0x62F7FC9C 0xB31028AA 0x34A5BE31 0x8A193BAC
R5:0x6D990BE6 0x4A4C5B16 0x62F7FC9C 0xB31028AA
I :0x96BE0F4C 0x5875CE62 0xD1179937 0xC9E04FE6
B :0xCF685F7F 0x0FC84888 0x78D47833 0xDE5DC1AE
I :0xD2771096 0xDF030D31 0xA94F72AB 0x06501D2C
R3:0xE42994F4 0x01F0E34A 0xA94F72AB 0x06501D2C
R3:0xA1248C59 0x85E78E06 0xE42994F4 0x01F0E34A
I :0x79CA37EC 0x20418FB2 0xA4BF133B 0xED4AE44E
B :0xB47E840B 0xFD00E8C4 0x6B34AB67 0x100BCB3A
I :0xA6826848 0xAA726BE5 0x1C64DBEE 0x895F718B
```

```
R5:0xE3DFED88 0x45AE01FF 0x1C64DBEE 0x895F718B
R5:0xC3192C59 0xEA3FF103 0xE3DFED88 0x45AE01FF
I :0x0DBA7E5B 0xC8C6F5B0 0xB53D3CE3 0x0C5BCE9E
B :0x3C5BCC92 0x40C1870C 0x7240B788 0xC45CB9A6
I :0x5301F112 0xE023FBA7 0x07B7ACE8 0xCCCF1D27
R3:0x8A48F64F 0x9CB676B1 0x07B7ACE8 0xCCCF1D27
R3:0xCAD86200 0xFB78A8DF 0x8A48F64F 0x9CB676B1
I :0xF8FE85FB 0x8AD014B7 0x970AC363 0x4F35C4BA
B :0x2F10C6B2 0x65E5502E 0xC2243A2F 0xA2019005
I :0x8829E5F6 0xD92401E6 0xFE154BBE 0xE3AE5450
R5:0x86C215CB 0xC234FC75 0xFE154BBE 0xE3AE5450
R5:0x33CAE854 0x849E683C 0x86C215CB 0xC234FC75
I :0xDF865177 0xCC8DB0B3 0x29BED9D9 0x235ECD0A
B :0x313EC90E 0xF6552C2C 0x0AADB211 0x19CBF595
I :0xBA5EF971 0x70951EEC 0x98A0BB2F 0xBB8F1684
R3:0x0A18287A 0xF53200CF 0x98A0BB2F 0xBB8F1684
R3:0x78CDCB6D 0x865DE218 0x0A18287A 0xF53200CF
I :0x238C2988 0xCB552594 0x18FAA2CB 0x4DCAB78D
B :0x79EAAFCF 0xE607BE95 0xD86164F6 0xECD91C1C
I :0xFAE4BAA3 0xBB72C4C0 0x60B9A4A5 0xC4B2AB32

CT=0xFAE4BAA3BB72C4C060B9A4A5C4B2AB32

Decrypt
I :0x79EAAFCF 0xE607BE95 0xD86164F6 0xECD91C1C
B :0x238C2988 0xCB552594 0x18FAA2CB 0x4DCAB78D
I :0x78CDCB6D 0x865DE218 0x0A18287A 0xF53200CF
R3:0x98A0BB2F 0xBB8F1684 0x0A18287A 0xF53200CF
R3:0xBA5EF971 0x70951EEC 0x98A0BB2F 0xBB8F1684
I :0x313EC90E 0xF6552C2C 0x0AADB211 0x19CBF595
B :0xDF865177 0xCC8DB0B3 0x29BED9D9 0x235ECD0A
I :0x33CAE854 0x849E683C 0x86C215CB 0xC234FC75
R5:0xFE154BBE 0xE3AE5450 0x86C215CB 0xC234FC75
R5:0x8829E5F6 0xD92401E6 0xFE154BBE 0xE3AE5450
I :0x2F10C6B2 0x65E5502E 0xC2243A2F 0xA2019005
B :0xF8FE85FB 0x8AD014B7 0x970AC363 0x4F35C4BA
I :0xCAD86200 0xFB78A8DF 0x8A48F64F 0x9CB676B1
R3:0x07B7ACE8 0xCCCF1D27 0x8A48F64F 0x9CB676B1
R3:0x5301F112 0xE023FBA7 0x07B7ACE8 0xCCCF1D27
I :0x3C5BCC92 0x40C1870C 0x7240B788 0xC45CB9A6
B :0x0DBA7E5B 0xC8C6F5B0 0xB53D3CE3 0x0C5BCE9E
I :0xC3192C59 0xEA3FF103 0xE3DFED88 0x45AE01FF
R5:0x1C64DBEE 0x895F718B 0xE3DFED88 0x45AE01FF
R5:0xA6826848 0xAA726BE5 0x1C64DBEE 0x895F718B
I :0xB47E840B 0xFD00E8C4 0x6B34AB67 0x100BCB3A
B :0x79CA37EC 0x20418FB2 0xA4BF133B 0xED4AE44E
I :0xA1248C59 0x85E78E06 0xE42994F4 0x01F0E34A
R3:0xA94F72AB 0x06501D2C 0xE42994F4 0x01F0E34A
R3:0xD2771096 0xDF030D31 0xA94F72AB 0x06501D2C
I :0xCF685F7F 0x0FC84888 0x78D47833 0xDE5DC1AE
B :0x96BE0F4C 0x5875CE62 0xD1179937 0xC9E04FE6
I :0x6D990BE6 0x4A4C5B16 0x62F7FC9C 0xB31028AA
R5:0x34A5BE31 0x8A193BAC 0x62F7FC9C 0xB31028AA
R5:0x89C7ACA9 0x805C112F 0x34A5BE31 0x8A193BAC
I :0x5F6BAEBB 0x7F238044 0xA5CDE028 0x7F30CB41
B :0x5A215E97 0x2511C596 0x005B7B29 0x05038ED3
I :0x00000000 0x00000000 0x00000000 0x00000000

PT=0x00000000000000000000000000000000
```