コンピュータセキュリティ 11－9
（2000．9．8）

次世代暗号Hierocryptの C言語による実装

佐野 文彦‡　村谷博文†　大熊建司†　川村信一†　本山雅彦†
†(株) 東芝 研究開発センター 〒212-8582 川崎市幸区小向東芝町1
{hirofumi.muratani, shinichi2.kawamura,masahiko.motoyama}@toshiba.co.jp
‡(株) 東芝 SI技術開発センター 〒183-8512 東京都府中市片町3-22
fumihiko.sano@toshiba.co.jp

あらまし　本稿では，Hierocrypt-L1およびHierocrypt-3をソフトウェア，ハードウェアおよびスマートカードに実装した場合のパフォーマンスを報告する．Hierocrypt-L1およびHierocrypt-3は広範なプラットフォームをカバーする共通鍵暗号アルゴリズムとして設計されている．64ビットブロック暗号アルゴリズムHierocrypt-L1はPentiumIII(550MHz)上のC言語実装で約140Mbpsを達成し，Z80(5MHz)上でも3.88msで暗号化が可能であり，MISTY-Iの従来の結果よりも高速であった．また，128ビットブロック暗号アルゴリズムHierocrypt-3は，PentiumIII(550MHz)上のアセンブラ言語実装で175Mbpsを達成し，Z80(5MHz)上でも10.03msで暗号化が可能となる．

Implementation of Hierocrypt

Fumihiko SANO‡　Hirofumi MURATANI†　Kenji OHKUMA†
Shinichi KAWAMURA†　Masahiko MOTOYAMA†
† Computer & Network Systems Laboratory, Coporate Research & Development Center
1, Komukai Toshiba-cho, Siwai-ku, Kawasaki-shi, 212-8582, Japan
{hirofumi.muratani, shinichi2.kawamura, masahiko.motoyama}@toshiba.co.jp
‡ Sytem Integration Technology Center
3-22, Katamachi, Fuchu-shi, Tokyo 183-8512, Japan
fumihiko.sano@toshiba.co.jp

Abstract　We report on the performance of Hierocrypt-L1 and Hierocrypt-3 on various platforms, such that software, hawrdware, and smartcard. They are designed to cover all over such platforms. The 64-bit block cipher Hierocrypt-L1 archives 140Mbits/sec on Pentium III(550MHz) using C language. The 128-bit block cipher Hierocrypt-3 archives 160Mbits/sec on Pentium III(550MHz) using assembly language. Each Hierocrypt-L1 and Hierocrypt-3 archives 3.88ms and 10.03ms on Z80(5MHz) using assembly language.

1　はじめに

本稿では，64ビット共通鍵ブロック暗号アルゴリズムHierocrypt-L1と128ビット共通鍵ブロック暗号アルゴリズムHierocrypt-3のソフトウェアおよびハードウェア実装効率について報告する．Hierocrypt-L1およびHierocrypt-3は入れ子SPN構造を採用した証明可能安全な共通鍵ブロック暗号である．Hierocrypt-L1はHierocryptの設計思想を従来の64ビットブロック暗号に適用した64ビットブロック暗号である．

Hierocryptはソフトウェアからハードウェアまでの広範なプラットフォームをカバーできる暗号アルゴリズムである．我々は，その性能を評価するために，Hierocrypt-L1とHierocrypt-3について，PC環境，スマートカード，ハードウェアのそれぞれについて実装を行った．

Hierocrypt-3の暗号化速度は，PenitumIII(550MHz)上において，C言語実装で約120Mbps，アセンブラ言語の実装で175Mbpsを達成した．また，Hierocrypt-L1の暗号化速度は，PenitumIII(550MHz)上において，C言語実装で約140Mbpsを達成し，所要サイクル数の比較ではMISTY-IのPentiumIIでの報告[2]から外挿した速度よりも高速である．さらに，Z80(12MHz)をCPUとするスマートカード環境においては，128bit鍵のHierocrypt-3は4.18msで1ブロックの暗号化処理を行うことができる．また，Hierocrypt-L1は同様の条件で1.62msで暗号化が可能であり，MISTY-Iよりも30%高速である．ハードウェア実装においては，スタンダードセルでの実装で，Hierocrypt-3は81.5Kゲートの回路規模で約900Mbps，Hierocrypt-L1は38.2Kゲートの回路規模で585Mbpsを達成した．

2　ソフトウェア実装

2.1　評価プラットフォーム仕様と速度評価方法

まず，評価に先立ち，表1に記述言語および開発環境を含むソフトウェアでの実装評価に用いたプラットフォームの詳細を記述する．まず速度評価の測定方法について簡単に述べる．本評価ではPentium IIIに内蔵されているTime Stamp Counter(TSC)を用いて，BENCH_COUNT回の鍵スケジュール部処理，データ暗号化処理およびデータ復号

表 1: 評価プラットフォーム仕様

機種	EQUIUM 5000 P55	
CPU	Penium III 550 MHz, CACHE(32KB), 2 次 CACHE(512KB)	
メモリ	256MB(SDRAM 100MHz)	
OS	Windows 2000 Profesional build 2195	
記述言語	C 言語	アセンブラ言語
開発環境	Visual C++ 6.0 SP3	MASM6.14
最適化	実行速度 (/O2)	—

処理のそれぞれの所要 CPU サイクル数を測定した．測定の曖昧さを避けるため，速度評価プログラム片を図 1 に示す．この (所要 CPU サイクル数/BENCH_COUNT) が関数呼び出しを含む 1 ブロックの処理に必要なサイクル数であり，処理スループットはこの値と CPU の駆動クロック数から求められる．

```
#define CPUID __asm __emit 0fh __asm __emit 0a2h
#define RDTSC __asm __emit 0fh __asm __emit 031h
  __asm {
    pushad
    CPUID
    RDTSC
    mov  cycles_high1, edx
    mov  cycles_low1, eax
    popad
  }
  for(i=0; i<BENCH_COUNT; i++)
     function_call(in, out, ekey); /* 評価対象 */
  __asm {
    pushad
    CPUID
    RDTSC
    mov  cycles_high2, edx
    mov  cycles_low2, eax
    popad
  }
  temp_cycles1 = ((unsigned __int64)cycles_high1 << 32) | cycles_low1;
  temp_cycles2 = ((unsigned __int64)cycles_high2 << 32) | cycles_low2;
  split = temp_cycle2 - temp_cycle1;
```

図 1: 速度評価プログラム片

2.2 メモリ量評価

メモリ使用量を表 2 に示す．暗号化および復号処理においては，段処理のループを展開していない場合の結果である．それぞれの処理において，C 言語記述で用いた 32 ビットワード変数をワークエリアの概算として見積もった．また，鍵スケジュール部および復号処理のワークエリアに記載されている値は拡大鍵格納領域の大きさを含む．

表 2: メモリ使用量

	Hierocrypt-L1			Hierocrypt-3			
処理名	コード量	テーブル量	ワーク	コード量	テーブル量	ワーク	備考
key scheduling	999	1,029	40 + 104	7,360	1,120	80 + 216	unroll
encryption	626	17,408	40	1,456	13,312	80	roll
decryption	1,187	17,408	40+104	2,419	13,312	80	roll
total	2,812	34,821	328	11,235	25,648	456	重複除く

(単位:バイト)

2.3 速度評価結果

各鍵長について 1,000,000 ブロックのデータに対して Hierocrypt-L1 および Hierocrypt-3 の ECB 処理所要サイクル数の測定を 10 回試行し，その最良値を表 3 に示す．実装は，C 言語あるいは MMX 拡張命令を含むアセンブラ言語で行った．表の MISTY-I[1] の数値は文献 [2] の Pentium II での結果を引用した．各処理のスループットはブロック幅とサイクル数から計算が可能である．

なお，表中の † で示す Hierocrypt-L1 の復号処理は，暗復同型タイプでの記述を行ったため暗号化処理用の拡大鍵から復号処理用の拡大鍵への変換処理を復号処理中に含んでおり，暗号化の約 2 倍のサイクル数を必要としている．この他に暗復非同型タイプ実装方法もあり，実装の最適化による速度向上の余地が見込める．

表 3: 速度評価 (サイクル数)

Algorithm	keylength	key scheduling	encrypt		decrypt		言語
			unroll	roll	unroll	roll	
Hierocrypt-3	128	370.0	600.5	615.0	(1012.0)*	1012.0	C
	192	386.5	710.0	722.5	(1251.0)*	1251.0	
	256	468.0	808.0	848.0	(1420.0)*	1420.0	
Hierocrypt-3	128	—	398.0		—		ASM
Hierocrypt-L1	128	179.0	253.0		521.0†		C
MISTY-I	128	—	184.0		—		C

(単位: cycles)
*:ループ展開による効果は認められなかった
†:暗復同型

3 スマートカード実装

Z80[4] を CPU として搭載しているスマートカードにおいて，処理速度優先の方針で Z80 アセンブラにより記述した実装結果を報告する．実装の対象として Z80 を搭載した東芝製スマートカード JT6N55[5] を使用し，実装は Z80 アセンブラ言語で行った．JT6N55 は 48KB の ROM，8KB の EEPROM，1KB の RAM を持つ．ただし，スマートカードではプログラムは ROM 領域に格納され改変は出来ない上，暗号処理以外の処理にも ROM，RAM および EEPROM が必要とされるため，要求されている速度が得られる範囲内で小さなコード量が求められる．なお，実装においては，処理速度優先としたが，スマートカードの全領域を使用するコードは現実的でないため，3KB 以内のコード量と on-the-fly 鍵生成を実装条件とした．

表 4 は，平文格納エリア，暗号文格納エリア，鍵格納エリアのアドレスをパラメータ指定し，Hierocrypt-L1 または Hierocrypt-3 のサブルーチンを呼び出して暗号文が暗号文格納エリア (RAM) までに必要となるステート数および必要となるメモリ量である．処理ステート数の見積もりには，4 ステートが最小インストラクションとなる，通常の Z80 アーキテクチャでの規定ステート数を用いた．処理時間の目安として，JT6N55 の駆動周波数 5MHz の場合と，高速な Z80 コアに相当する 12MHz の場合を表に示した．本評価においては，最も処理効率の良い 1,280 バイトの参照テーブルを使用したため，コード量はコンパクトな実装と比較して大きい．コンパクトな実装の場合には，参照テーブルは S-box の 256 バイトと若干の定数だけが必要であり，コード量を約 1KB 削減できる．

表 4: スマートカードでの速度およびメモリ量評価

Algorithm	key length	RAM	ROM	encryption		
	(bits)	(bytes)	(bytes)	(states)	(@5MHz)	(@12MHz)
Hierocrypt-3	128	58	2,889	50,157	10.03ms	4.18ms
Hierocrypt-L1	128	26	2,447	19,399	3.88ms	1.62ms
MISTY-I	128	44	1,598	25,486	5.10ms	2.12ms

4 ハードウェア実装

ハードウェア実装として，スタンダードセルと FPGA による実装結果を報告する．スタンダードセルの使用プロセスは 0.14 μ m 3 層配線 CMOS であり，設計環境として，SYNOPSYS 社製 Design Compier 1999.10-3 を使用した．シミュレーション条件にはコマーシャル用ワーストケースの 1.35V 70 ℃ (標準ケースでは，1.5V 25 ℃) を採用した．

また，FPGA の設計には ALTERA 社製 Max+plus II ver. 9.6 を，リソースとして ALTERA 社製 Flex 10K ファミリーを使用した．それぞれの場合に付いて，シミュレーション結果を以下に示す．

表 5: ハードウェア実装結果

アルゴリズム	実装方法	オプション	ラウンド数 (クロック数)	クロック周波数	データスループット	ゲート数	ロジックセル数
Hierocrypt-L1 (128bit key)	SC	速度優先	6(14)	128.2 MHz	586 Mbps	38.2K	-
	FPGA	速度優先	6(14)	11.16 MHz	51.0 Mbps	-	11K
Hierocrypt-3 (256bit key)	SC	速度優先	8(18)	126.1 MHz	897 Mbps	81.5K	-
	SC	面積優先	8(280)	185.1 MHz	84.6 Mbps	26.7K	-
	FPGA	速度優先	8(18)	7.39 MHz	52.6 Mbps	-	11K
	FPGA	面積優先	8(280)	8.95 MHz	4.1 Mbps	-	6.3K

5 まとめ

本稿では，証明可能安全な共通鍵ブロック暗号 Hierocrypt のうち，64 ビットブロック暗号 Hierocrypt-L1 および 128 ビットブロック暗号 Hierocrypt-3 のソフトウェア，スマートカード，ハードウェアのそれぞれの環境での実装結

果を報告した．Hierocrypt–L1 および Hierocrypt–3 は，各プラットフォームにおいて優秀なパフォーマンスを達成し，このことからも，Hierocrypt–L1 および Hierocrypt–3 は広範はプラットフォームをカバーできる柔軟な設計の暗号アルゴリズムであると言える．なお，紙面の都合上，実装結果の一部や高速化に使ったさまざまな手法を説明することは出来なかったが，別の機会において発表したい．最後に，参考のために本稿の末尾に Hierocrypt–L1 および Hierocrypt–3 のサンプル実装とテストデータを付録として加える．

参考文献

[1] M. Matsui, *"New Block Encryption Algorithm MISTY"*, Fast Software Encryption, 4th International Workshop Proceeding, LNCS **1267**, Springer-Verlag, 1997, pp.54-68.

[2] 中嶋純子, 松井充, *"MISTY のソフトウエアによる高速実装法について (II)"*, 情報とセキュリティシンポジウム, SCIS'98-9.1.B, 1998.

[3] F. Sano, M. Koike, S. Kawamura, and M. Shiba, *"Performance Evaluation of AES Finalists on the High-End Smart Card"*, 3rd AES Conference, 2000.

[4] ZiLOG, *"Z80 Microprocessor Products"*, available on http://www.zilog.com/products/z80.html

[5] JT6N55, http://www.toshiba.co.jp/about/press/2000_02/pr_j1801.htm

Hierocrypt–L1 サンプル実装

```
/*****************************************
 *   64-bit block cipher Hierocrypt-L1    *
 *          Version 1.0 July 10           *
 *   Copyright Toshiba Corporation 2000   *
 *****************************************/
#include <stdio.h>
#include <stdlib.h>
typedef unsigned char   BYTE;
typedef unsigned int    POLY32;
#define MAX_ROUND       6
#define primitive       0x163
/*****************************************************************/
BYTE sbox[256] = {
  7,252, 85,112,152,142,132, 78,188,117,206, 24,  2,233, 93,128, 28, 95,120, 66,
157, 46,245,232,198,122, 47,164,178, 95, 25,135, 11,155,156,211,195,119, 61,111,
185, 45, 77,247,140,167,172, 23, 60, 90, 65,201, 41,237,222, 39,105, 48,114,168,
149, 62,249,216, 33,139, 68,215, 17, 13, 72,253,106,  1, 87,229,189,133,236, 30,
 55,159,181,154,124,  9,241,177,148,129,130,  8,251,192, 81, 15, 97,127, 26, 86,
150, 19,193,103,153,  3, 94,182,202,250,158,223,214,131,204,162, 18, 35,183,101,
208, 57,125, 59,213,176,175, 31,  6,200, 52,197, 27,121, 75,102,191,136, 74,196,
239, 88, 63, 10, 44,115,209,248,107,230, 32,184, 34, 67,179, 51,231,240,113,126,
 82,137, 71, 99, 14,109,227,190, 89,100,238,246, 56, 92,244, 91, 73,212,224,243,
187, 84, 38, 43,  0,134,144,255,254,166,123,  5,173,104,161, 16,235,199,226,242,
 70,138,108, 20,110,207, 53, 69, 80,210,148,116,147,225,218,174,169, 83,228, 64,
205,186,151,163,145, 49, 37,118, 54, 50, 40, 58, 36, 76,219,217,141,220, 98, 42,
234, 21,221,194,165, 12,  4, 29,143,203,180, 79, 22,171,170,160} ;
BYTE isbox[256] = {
184, 73, 12,105,246,191,128,  0, 91, 85,143, 32,245, 69,164, 95,195, 68,116,101,
203,241,252, 47, 11, 30, 98,132, 16,247, 79,127,150, 64,152,117,232,226,182, 55,
230, 52,239,183,144, 41, 21, 26, 57,225,229,155,130,206,228, 80,172,121,231,123,
 48, 38, 61,142,219, 50, 19,153, 66,207,200,162, 70,176,138,134,233, 42,  7,251,
208, 94,160,217,181,  2, 99, 74,141,168, 49,175,173, 14,108, 29, 17, 96,238,163,
169,119,135,103,193, 56, 72,148,202,165,204, 39,  3,158, 58,145,211,  9,227, 37,
 18,133, 25,190, 84,122,159, 97, 15, 89, 90,113,  6, 77,185, 31,137,161,201, 65,
 44,236,  5,246,186,224,210,212, 88, 60,100,222,  4,104, 83, 33, 34, 20,110, 81,
255,194,115,223, 27,244,189, 46, 59,218,254,253, 45,192,215,126,125, 87, 28,154,
250, 82,107,118,151, 40,221,180,  8, 75,167,136, 93,102,243, 35,139,131, 24,197,
129, 51,108,249,114,220, 10,205,120,146,209, 35,177,124,112, 67, 63,235,214,234,
237,242, 54,111,178,213,198,166,218, 75,149,156, 23, 13,240,196, 78, 53,170,140,
157, 86,199,179,174, 22,171, 43,147, 62,109, 92,  1, 71,188,187};
BYTE hconst[5][4] = {0x5a, 0x82, 0x79, 0x99, 0x6e, 0xd9, 0xeb, 0xa1, 0x8f, 0x1b,
                     0xbc, 0xdc, 0xca, 0x62, 0xc1, 0xd6, 0x8a, 0xf5, 0xda, 0xf7};
int hindex[] = {1,2,3,4,4,3,2};
BYTE mds[4][4] = {{0xc4, 0x65, 0xc8, 0x8b},  {0x8b, 0xc4, 0x65, 0xc8},
                  {0xc8, 0x8b, 0xc4, 0x65},  {0x65, 0xc8, 0x8b, 0xc4}};
BYTE mds_inv[4][4] = {{0x82, 0xc4, 0x34, 0xf6},  {0xf6, 0x82, 0xc4, 0x34},
                      {0x34, 0xf6, 0x82, 0xc4},  {0xc4, 0x34, 0xf6, 0x82}};
int     mdsh[2][2] = {{0x5,0x7},{0xa,0xb}};
int     mdsh_inv[4][4] = {{0xc,0xa},{0x5,0xb}};
int poly32_deg(POLY32 a)
{
  int n = -1;
  for(; a; a >>= 1) n++;
  return n;
}
POLY32 poly32_mul(POLY32 a, POLY32 b)
{
  POLY32 c = 0;
  for(; b; b >>= 1, a <<= 1) if(b&1) c ^= a;
  return c;
}
POLY32 poly32_mod(POLY32 a, POLY32 b)
{
  int da = poly32_deg(a);
  int db = poly32_deg(b);
  POLY32 t;
  if(da < db) return a;
  if(da == db) return a^b;
  b <<= da - db;
  for(t = 1<<da; da >= db; da--){
    if(a&t) a ^= b; b >>= 1; t >>= 1;
  }
  return a;
}
void hcryptLi_mdsl(BYTE *in, BYTE *out)
{
  int   i, j;
  POLY32        m;
  for(i=0; i<4; i++) {
    for(m=j=0; j<4; j++)
      m ^= poly32_mod(poly32_mul(mds[i][j], in[j]), primitive);
    out[i] = m;
  }
}
void hcryptLi_xs(BYTE *in, BYTE *out, BYTE *k1, BYTE *k2)
{
  BYTE  t[4], u[4];
  int           i;
  for(i=0; i<4; i++)        u[i] = in[i]^k1[i];     /* key XOR */
  for(i=0; i<4; i++)        t[i] = sbox[u[i]];      /* sbox */
  hcryptLi_mdsl(t, u);              /* MDS_L */
  for(i=0; i<4; i++)        t[i] = u[i]^k2[i];      /* key XOR */
  for(i=0; i<4; i++)        out[i] = sbox[t[i]];    /* sbox */
}
void mdsh_mul(BYTE *in, BYTE *out, int x)
{
  int   i;
  BYTE  u[4];
  for(i=0; i<4; i++)        u[i] = 0;
  if (x&1) {u[0]^=in[0];        u[1]^=in[1]; u[2]^=in[2]; u[3]^=in[3];}
  if (x&2) {u[0]^=in[1];        u[1]^=in[2]; u[2]^=in[3]^in[0]; u[3]^=in[0];}
  if (x&4) {u[0]^=in[2];        u[1]^=in[3]^in[0]; u[2]^=in[0]^in[1]; u[3]^=in[1];}
  if (x&8) {u[0]^=in[0]^in[3]; u[1]^=in[1]^in[0]; u[2]^=in[2]^in[1]; u[3]^=in[2];}
  for(i=0; i<4; i++) out[i] = u[i];
}
void hcryptLi_mdsh(BYTE in[2][4], BYTE out[2][4])
{
  int           i, j, k;
  BYTE  tmp[4];
  for(i=0; i<2; i++)
    for(j=0; j<4; j++)    out[i][j] = 0;
  for(i=0; i<2; i++) {
    for(j=0; j<2; j++) {
      mdsh_mul(&in[j][0], tmp, mdsh[i][j]);
      for(k=0; k<4; k++)    out[i][k] ^= tmp[k];
    }
  }
}
void hcryptLi_encrypt(BYTE *in, BYTE *out, BYTE ks[MAX_ROUND+1][4][4])
{
  BYTE  t[2][4], u[2][4];
  int i, j, r, n = MAX_ROUND-1;
  for(r=i=0; i<2; i++)
    for(j=0; j<4; j++, r++)             t[i][j] = in[r];
  for(r=0; r<n; r++) {
    hcryptLi_xs(&t[0][0], &u[0][0], &ks[r][0][0], &ks[r][2][0]);
    hcryptLi_xs(&t[1][0], &u[1][0], &ks[r][1][0], &ks[r][3][0]);
    hcryptLi_mdsh(u, t);
  }
  hcryptLi_xs(&t[0][0], &u[0][0], &ks[n][0][0], &ks[n][2][0]);
  hcryptLi_xs(&t[1][0], &u[1][0], &ks[n][1][0], &ks[n][3][0]);
  for(r=i=0; i<2; i++)
    for(j=0; j<4; j++, r++)     out[r] = u[i][j]^ks[n+1][i][j];
}
void hcryptLi_imdsl(BYTE *in, BYTE *out)
{
  int   i, j;
  POLY32        m;
  for(i=0; i<4; i++) {
    for(m=j=0; j<4; j++)
      m ^= poly32_mod(poly32_mul(mds_inv[i][j], in[j]), primitive);
    out[i] = m;
  }
}
void hcryptLi_ixs(BYTE *in, BYTE *out, BYTE *k1, BYTE *k2)
{
  BYTE  t[4], u[4];
  int           i;
  for(i=0; i<4; i++)        u[i] = in[i]^k1[i];     /* key XOR */
  for(i=0; i<4; i++)        t[i] = isbox[u[i]];     /* sbox */
  hcryptLi_imdsl(t, u);             /* MDS_L */
  for(i=0; i<4; i++)        t[i] = u[i]^k2[i];      /* key XOR */
  for(i=0; i<4; i++)        out[i] = isbox[t[i]];   /* sbox */
```

```
}
void hcryptLi_imdsh(BYTE in[2][4], BYTE out[2][4])
{
  int           i, j, k;
  BYTE  tmp[4];
  for(i=0; i<4; i++)
    for(j=0; j<4; j++)       out[i][j] = 0;
  for(i=0; i<2; i++) {
    for(j=0; j<2; j++) {
      mdsh_mul(&in[j][0], tmp, mdsh_inv[i][j]);
      for(k=0; k<4; k++)         out[i][k] ^= tmp[k];
    }
  }
}
void hcryptLi_decrypt(BYTE *in, BYTE *out, BYTE ks[MAX_ROUND+1][4][4])
{
  BYTE  t[4][4], u[4][4];
  int i, j, r, n=5;
  for(r=i=0; i<2; i++)
    for(j=0; j<4; j++, r++)       t[i][j] = in[r];
  for(r=0; r<n; r++) {
    hcryptLi_ixs(&t[0][0], &u[0][0], &ks[r][0][0], &ks[r][2][0]);
    hcryptLi_ixs(&t[1][0], &u[1][0], &ks[r][1][0], &ks[r][3][0]);
    hcryptLi_imdsh(u, t);
  }
  hcryptLi_ixs(&t[0][0], &u[0][0], &ks[n][0][0], &ks[n][2][0]);
  hcryptLi_ixs(&t[1][0], &u[1][0], &ks[n][1][0], &ks[n][3][0]);
  for(r=i=0; i<2; i++)
    for(j=0; j<4; j++, r++)       out[r] = u[i][j]^ks[n+1][i][j];
}
void hcryptLi_keyf(BYTE *in, BYTE *fout, BYTE *fkey)
{
  int           i;
  for(i=0; i<4; i++)         fout[i] = sbox[in[i]^fkey[i]];
  fout[0] ^= fout[2];  fout[1] ^= fout[3];
  fout[2] ^= fout[1];  fout[3] ^= fout[0];
}
void swap_key(BYTE *1, BYTE *r)
{
  BYTE  t;
  int   i;
  for(i=0; i<4; i++) {t = l[i]; l[i] = r[i]; r[i] = t;}
}
void hcryptLi_keyp(BYTE k[4][4], BYTE kout[4][4], int index)
{
  int   i;
  BYTE  fout[4];
  for(i=0; i<2; i++) {k[2][i] ^= k[3][i]; k[2][i+2] ^= k[3][i+2];}  /* P(16) */
  for(i=0; i<2; i++) {k[3][i] ^= k[2][i+2]; k[3][i+2] ^= k[2][i];}  /* P(16) */
  mdsh_mul(&k[2][0], &k[2][0], 5);
  mdsh_mul(&k[3][0], &k[3][0], 0xb);
  for(i=0; i<4; i++)         k[2][i] ^= hconst[index][i];
  hcryptLi_keyf(&k[i][0], fout, &k[2][0]);
  for(i=0; i<4; i++)         k[0][i] ^= fout[i];              /* L+f(R) */
  for(i=0; i<4; i++)         kout[0][i] = k[0][i];
  for(i=0; i<4; i++)         kout[1][i] = fout[i  ]^k[2][i  ];
  for(i=0; i<4; i++)         kout[2][i] = fout[i  ]^k[3][i  ];
  for(i=0; i<4; i++)        kout[3][i] = k[1][i  ]^k[3][i  ];
  swap_key(&k[0][0], &k[1][0]);
}
void hcryptLi_keyc(BYTE k[4][4], BYTE kout[4][4], int index)
{
  int           i;
  BYTE  fout[4];
  swap_key(&k[0][0], &k[1][0]);
  hcryptLi_keyf(&k[1][0], fout, &k[2][0]);
  for(i=0; i<4; i++)         k[0][i] ^= fout[i];              /* L+f(R) */
  for(i=0; i<4; i++)         kout[0][i] = k[0][i]^k[2][i  ];
  for(i=0; i<4; i++)         k[2][i] ^= hconst[index][i];
  mdsh_mul(&k[2][0], &k[2][0], 0xb);
  mdsh_mul(&k[3][0], &k[3][0], 0x5);
  for(i=0; i<4; i++)         kout[1][i] = fout[i  ]^k[2][i  ];
  for(i=0; i<4; i++)         kout[2][i] = fout[i  ]^k[3][i  ];
  for(i=0; i<4; i++)         kout[3][i] = k[1][i  ]^k[3][i  ];
  for(i=0; i<2; i++) {k[3][i] ^= k[2][i+2]; k[3][i+2] ^= k[2][i];}  /* (P(16))^-1 */
  for(i=0; i<2; i++) {k[2][i] ^= k[3][i]; k[2][i+2] ^= k[3][i+2];}  /* (P(16))^-1 */
}
void hcryptLi_setkey(BYTE *key, BYTE ks[MAX_ROUND+1][4][4])
{
  BYTE  k[4][4], fout[4];
  int   i, j, pos, r;
  /* copy & padding */
  for(pos=i=0; i<4; i++)
    for(j=0; j<4; j++)       k[i][j] = key[pos++];
  mdsh_mul(&k[2][0], &k[2][0], 5);  /* dummy start */
  mdsh_mul(&k[3][0], &k[3][0], 0xb);
  for(i=0; i<4; i++)         k[2][i] ^= hconst[0][i];
  hcryptLi_keyf(&k[i][0], fout, &k[2][0]);
  for(i=0; i<4; i++)         k[0][i] ^= fout[i];              /* L+f(R) */
  swap_key(&k[0][0], &k[i][0]);  /* dummy end */
  for(r=0; r<4; r++)         hcryptLi_keyp(&k[0], &ks[r][0], hindex[r]);
  for(r=4; r<7; r++)         hcryptLi_keyc(&k[0], &ks[r][0], hindex[r]);
}
void hcryptLi_key_edtrans(BYTE eks[MAX_ROUND+1][4][4], BYTE dks[MAX_ROUND+1][4][4])
{
  int   i, j, k, r = MAX_ROUND;
  for(j=0; j<2; j++)
    for(k=0; k<4; k++)       dks[0][j][k] = eks[r][j][k];
  for(i=1; i<r; i++) {
    for(j=2; j<4; j++)       hcryptLi_imdsl(&eks[r-i][j][0], &dks[i-1][j][0]);
    hcryptLi_imdsh(&eks[r-i][0], &dks[i][0]);
  }
  for(j=2; j<4; j++)         hcryptLi_imdsl(&eks[0][j][0], &dks[r-1][j][0]);
  for(j=0; j<2; j++)
    for(k=0; k<4; k++)        dks[r][j][k] = eks[0][j][k];
}
int main()
{
  BYTE key[16], in[8], out[8], res[8];
  BYTE  ks[MAX_ROUND+1][4][4], ks2[MAX_ROUND+1][4][4];
  hcryptLi_setkey(key, ks);                          /* 拡大鍵生成 */
  hcryptLi_encrypt(in, out, ks);         /* 暗号化 */
  hcryptLi_key_edtrans(ks, ks2);         /* 復号用鍵生成 (ks->ks2 変換) */
  hcryptLi_decrypt(out, res, ks2);       /* 復号 */
  return EXIT_SUCCESS;
}
```

Hierocrypt-3 サンプル実装

```
/*****************************************
 *   128-bit block cipher Hierocrypt-3    *
 *            Version 1.0 July 10         *
 *   Copyright Toshiba Corporation 2000   *
 *****************************************/
#include <stdio.h>
#include <stdlib.h>
typedef unsigned char       BYTE;
typedef unsigned int    POLY32;
#define MAX_ROUND   8
#define primitive 0x163
BYTE sbox[256] = {
  7,252, 85,112,152,142,132, 78,188,117,206, 24,  2,233, 93,128, 28, 96,120, 66,
157, 46,245,232,198,122, 47,164,178, 95, 25,135, 11,155,156,211,195,119, 61,111,
185, 45, 77,247,140,167,172, 23, 60, 90, 65,201, 41,237,222, 39,105, 48,114,168,
149, 62,249,216, 33,139, 68,215, 17, 13, 72,253,106,  1, 87,229,189,133,236, 30,
 55,159,181,154,124,  9,241,177,148,129,130,  8,251,192, 81, 15, 97,127, 26, 86,
150, 19,193,103,153,  3, 94,182,202,250,158,223,214,131,204,162, 18, 35,183,101,
208, 57,125, 59,213,176,175, 31,  6,200, 52,197, 27,121, 75,102,191,136, 74,196,
239, 88, 63, 10, 44,115,209,248,107,230, 32,184, 34, 67,179, 51,231,240,113,126,
 82,137, 71, 99, 14,109,227,190, 89,100,238,246, 56, 92,244, 91, 73,212,224,243,
187, 84, 38, 43,  0,134,144,255,254,166,123,  5,173,104,161, 16,235,199,226,242,
 70,138,108, 20,110,207, 53, 69, 80,210,146,116,147,225,218,174,169, 83,228, 64,
205,186,151,163,145, 49, 37,118, 54, 50, 40, 58, 36, 76,219,217,141,220, 98, 42,
234, 21,221,194,165, 12,  4, 29,143,203,180, 79, 22,171,170,160} ;
BYTE isbox[256] = {
184, 73, 12,105,246,191,128,  0, 91, 85,143, 32,245, 69,164, 95,195, 68,116,101,
203,241,252, 47, 11, 30, 98,132, 16,247, 79,127,150, 64,152,117,232,226,182, 55,
230, 52,239,183,144, 41, 21, 26, 57,225,229,155,130,206,228, 80,172,121,231,123,
 48, 38, 61,142,219, 50, 19,153, 66,207,200,162, 70,176,138,134,233, 42,  7,251,
208, 94,160,217,181,  2, 99, 74,141,168, 49,175,173, 14,106, 29, 17, 96,238,163,
169,119,135,103,193, 56, 72,148,202,165,204, 39,  3,158, 58,145,211,  9,227, 37,
 18,133, 25,190, 84,122,159, 97, 15, 89, 90,113,  6, 77,185, 31,137,161,201, 65,
 44,236,  5,248,186,224,210,212, 88, 60,100,222,  4,104, 83, 33, 34, 20,110, 81,
255,194,115,223, 27,244,189, 45, 59,216,254,253, 46,192,215,126,125, 87, 28,154,
250, 82,107,118,151, 40,221,180,  8, 76,167,136, 93,102,243, 36,139,131, 24,197,
129, 51,108,249,114,220, 10,205,120,146,209, 35,177,124,112, 67, 63,235,214,234,
237,242, 54,111,178,213,198,166,218, 75,149,156, 23, 13,240,196, 78, 53,170,140,
157, 86,199,179,174, 22,171, 43,147, 62,109, 92,  1, 71,188,187};
BYTE hconst[4][4] = {0x5a, 0x82, 0x79, 0x99, 0x6e, 0xd9, 0xeb, 0xa1,
                     0x8f, 0x1b, 0xbc, 0xdc, 0xca, 0x62, 0xc1, 0xd6};
int gindex[6][2] = {3,0,2,1,1,3,0,2,2,3,1,0};
int kconst[3][10] = {{0, 1, 2, 3, 3, 2, 1,-1,-1,-1},
  {1, 0, 3, 2, 2, 3, 0, 1,-1,-1},{4, 0, 2, 1, 3, 3, 1, 2, 0,-1}};
BYTE mds[4][4] = {{0xc4, 0x65, 0xc8, 0x8b},{0x8b, 0xc4, 0x65, 0xc8},
  {0xc8, 0x8b, 0xc4, 0x65},{0x65, 0xc8, 0x8b, 0xc4}};
BYTE mds_inv[4][4] = {{0x82, 0xc4, 0x34, 0xf6},{0xf6, 0x82, 0xc4, 0x34},
  {0x34, 0xf6, 0x82, 0xc4},{0xc4, 0x34, 0xf6, 0x82}};
int mdsh[4][4] = {{0x5,0x5,0xa,0xe},{0xe,0x5,0x5,0xa},
  {0xa,0xe,0x5,0x5},{0x5,0xa,0xe,0x5}};
int mdsh_inv[4][4] = {{0xb,0xe,0xe,0xd},{0xd,0xB,0xe,0xe},
  {0xe,0xd,0xb,0xe},{0xe,0xe,0x6,0xb}};
int poly32_deg(POLY32 a)
{
  int n = -1;
  for(; a; a >>= 1) n++;
  return n;
}
POLY32 poly32_mul(POLY32 a, POLY32 b)
{
  POLY32 c = 0;
  for(; b; b >>= 1, a <<= 1) if(b&1) c ^= a;
  return c;
}
POLY32 poly32_mod(POLY32 a, POLY32 b)
{
  int da = poly32_deg(a);
  int db = poly32_deg(b);
  POLY32 t;
  if(da < db) return a;
  if(da == db) return a^b;
  b <<= da - db;
  for(t = 1<<da; da >= db; da--){
    if(a&t) a ^= b; b >>= 1; t >>= 1;
  }
  return a;
}
void hcrypt_mdsl(BYTE *in, BYTE *out)
{
  int   i, j;
  POLY32    m;
  for(i=0; i<4; i++) {
    m = 0;
    for(j=0; j<4; j++) m ^= poly32_mod(poly32_mul(mds[i][j], in[j]), primitive);
    out[i] = m;
  }
}
void hcrypt_xs(BYTE *in, BYTE *out, BYTE *k1, BYTE *k2)
{
  BYTE  t[4], u[4];
  int       i;
  for(i=0; i<4; i++)    u[i] = in[i]^k1[i]; /* key XOR */
  for(i=0; i<4; i++)    t[i] = sbox[u[i]];  /* sbox */
  hcrypt_mdsl(t, u);            /* MDS_L */
  for(i=0; i<4; i++)    t[i] = u[i]^k2[i];  /* key XOR */
  for(i=0; i<4; i++)    out[i] = sbox[t[i]];    /* sbox */
}
void mdsh_mul(BYTE *in, BYTE *out, int x)
{
  int   i;
  BYTE  u[4];
  for(i=0; i<4; i++)        u[i] = 0;
  if (x&1) {u[0]^=in[0];       u[1]^=in[1]; u[2]^=in[2]; u[3]^=in[3];}
  if (x&2) {u[0]^=in[1];       u[1]^=in[2]; u[2]^=in[3]^in[0]; u[3]^=in[0];}
  if (x&4) {u[0]^=in[2];       u[1]^=in[3]^in[0]; u[2]^=in[0]^in[1]; u[3]^=in[1];}
  if (x&8) {u[0]^=in[0]^in[3]; u[1]^=in[1]^in[0]; u[2]^=in[2]^in[1]; u[3]^=in[2];}
  for(i=0; i<4; i++) out[i] = u[i];
```

```
}
void hcrypt_mdsh(BYTE in[4][4], BYTE out[4][4])
{
  int       i, j, k;
  BYTE  tmp[4];
  for(i=0; i<4; i++)
    for(j=0; j<4; j++)    out[i][j] = 0;
  for(i=0; i<4; i++) {
    for(j=0; j<4; j++) {
      mdsh_mul(&in[j][0], tmp, mdsh[i][j]);
      for(k=0; k<4; k++)        out[i][k] ^= tmp[k];
    }
  }
}
void hcrypt_encrypt(BYTE *in, BYTE *out, BYTE ks[MAX_ROUND+1][8][4], int key_len)
{
  BYTE  t[4][4], u[4][4];
  int i, j, r,n;
  for(r=i=0; i<4; i++)
    for(j=0; j<4; j++, r++)
      t[i][j] = in[r];
  n = key_len/64+3;
  for(r=0; r<n; r++) {
    hcrypt_xs(&t[0][0], &u[0][0], &ks[r][0][0], &ks[r][4][0]);
    hcrypt_xs(&t[1][0], &u[1][0], &ks[r][1][0], &ks[r][5][0]);
    hcrypt_xs(&t[2][0], &u[2][0], &ks[r][2][0], &ks[r][6][0]);
    hcrypt_xs(&t[3][0], &u[3][0], &ks[r][3][0], &ks[r][7][0]);
    hcrypt_mdsh(u, t);
  }
  hcrypt_xs(&t[0][0], &u[0][0], &ks[n][0][0], &ks[n][4][0]);
  hcrypt_xs(&t[1][0], &u[1][0], &ks[n][1][0], &ks[n][5][0]);
  hcrypt_xs(&t[2][0], &u[2][0], &ks[n][2][0], &ks[n][6][0]);
  hcrypt_xs(&t[3][0], &u[3][0], &ks[n][3][0], &ks[n][7][0]);
  for(r=i=0; i<4; i++)
    for(j=0; j<4; j++, r++)
      out[r] = u[i][j]^ks[n+1][i][j];
}
void hcrypt_imds1(BYTE *in, BYTE *out)
{
  int   i, j;
  POLY32    m;
  for(i=0; i<4; i++) {
    m = 0;
    for(j=0; j<4; j++)    m ^= poly32_mod(poly32_mul(mds_inv[i][j], in[j]), primitive);
    out[i] = m;
  }
}
void hcrypt_ixs(BYTE *in, BYTE *out, BYTE *ki, BYTE *k2)
{
  BYTE  t[4], u[4];
  int       i;
  for(i=0; i<4; i++)    u[i] = in[i]^ki[i]; /* key XOR */
  for(i=0; i<4; i++)    t[i] = isbox[u[i]]; /* sbox */
  hcrypt_imds1(t, u);           /* MDS_L */
  for(i=0; i<4; i++)    t[i] = u[i]^k2[i];  /* key XOR */
  for(i=0; i<4; i++)    out[i] = isbox[t[i]];   /* sbox */
}
void hcrypt_imdsh(BYTE in[4][4], BYTE out[4][4])
{
  int       i, j, k;
  BYTE  tmp[4];
  for(i=0; i<4; i++)
    for(j=0; j<4; j++)    out[i][j] = 0;
  for(i=0; i<4; i++) {
    for(j=0; j<4; j++) {
      mdsh_mul(&in[j][0], tmp, mdsh_inv[i][j]);
      for(k=0; k<4; k++)        out[i][k] ^= tmp[k];
    }
  }
}
void hcrypt_decrypt(BYTE *in, BYTE *out, BYTE ks[MAX_ROUND+1][8][4], int key_len)
{
  BYTE  t[4][4], u[4][4];
  int i, j, r, n;
  for(r=i=0; i<4; i++)
    for(j=0; j<4; j++, r++)   t[i][j] = in[r];
  n = key_len/64+3;
  for(r=0; r<n; r++) {
    hcrypt_ixs(&t[0][0], &u[0][0], &ks[r][0][0], &ks[r][4][0]);
    hcrypt_ixs(&t[1][0], &u[1][0], &ks[r][1][0], &ks[r][5][0]);
    hcrypt_ixs(&t[2][0], &u[2][0], &ks[r][2][0], &ks[r][6][0]);
    hcrypt_ixs(&t[3][0], &u[3][0], &ks[r][3][0], &ks[r][7][0]);
    hcrypt_imdsh(u, t);
  }
  hcrypt_ixs(&t[0][0], &u[0][0], &ks[r][0][0], &ks[n][4][0]);
  hcrypt_ixs(&t[1][0], &u[1][0], &ks[r][1][0], &ks[n][5][0]);
  hcrypt_ixs(&t[2][0], &u[2][0], &ks[r][2][0], &ks[n][6][0]);
  hcrypt_ixs(&t[3][0], &u[3][0], &ks[r][3][0], &ks[n][7][0]);
  for(r=i=0; i<4; i++)
    for(j=0; j<4; j++, r++)   out[r] = u[i][j]^ks[n+1][i][j];
}
void hcrypt_keyf(BYTE *in, BYTE *fout, BYTE *fkey)
{
  int       i;
  for(i=0; i<8; i++)    fout[i] = sbox[in[i]^fkey[i]];
  fout[0] ^= fout[4]; fout[1] ^= fout[5]; fout[2] ^= fout[6]; fout[3] ^= fout[7];
  fout[4] ^= fout[2]; fout[5] ^= fout[3]; fout[6] ^= fout[0]; fout[7] ^= fout[1];
}
void swap_key(BYTE *l, BYTE *r)
{
  BYTE  t;
  int   i;
  for(i=0; i<8; i++) {t = l[i]; l[i] = r[i]; r[i] = t;}
}
void hcrypt_keyp(BYTE k[4][8], BYTE kout[8][4], int index)
{
  int   i;
  BYTE  fout[8];
  for(i=0; i<4; i++) { k[2][i] ^= k[3][i]; k[2][i+4] ^= k[3][i+4];  }/* P(32) */
  for(i=0; i<4; i++) { k[3][i] ^= k[2][i+4]; k[3][i+4] ^= k[2][i];  }/* P(32) */
  mdsh_mul(&k[2][0], &k[2][0], 5);
  mdsh_mul(&k[2][4], &k[2][4], 0xe);
  mdsh_mul(&k[3][0], &k[3][0], 5);
  mdsh_mul(&k[3][4], &k[3][4], 0xe);
  for(i=0; i<4; i++)    k[2][i] ^= hconst[gindex[index][0]][i];
  for(i=0; i<4; i++)    k[2][i+4] ^= hconst[gindex[index][1]][i];
  hcrypt_keyf(&k[1][0], fout, &k[2][0]);
  for(i=0; i<8; i++)    k[0][i] ^= fout[i];     /* L xor f(R) */
  for(i=0; i<4; i++) {kout[0][i] = k[0][i];   kout[1][i] = k[0][i+4];}
  for(i=0; i<4; i++) {kout[2][i] = fout[i]^k[2][i]; kout[3][i] = fout[i+4]^k[2][i+4];}
  for(i=0; i<4; i++) {kout[4][i] = fout[i]^k[3][i]; kout[5][i] = fout[i+4]^k[3][i+4];}
  for(i=0; i<4; i++) {kout[6][i] = k[i][i]  ^k[3][i]; kout[7][i] = k[i][i+4]^k[3][i+4];}
  swap_key(&k[0][0], &k[1][0]);
}
void hcrypt_keyc(BYTE k[4][8], BYTE kout[8][4], int index)
{
  int   i;
  BYTE  fout[8];
  swap_key(&k[0][0], &k[1][0]);
  hcrypt_keyf(&k[1][0], fout, &k[2][0]);
  for(i=0; i<8; i++)    k[0][i] ^= fout[i];     /* L+f(R) */
  for(i=0; i<4; i++) {kout[0][i] = k[0][i]^k[2][i]; kout[1][i] = k[0][i+4]^k[2][i+4];}
  for(i=0; i<4; i++)    k[2][i] ^= hconst[gindex[index][0]][i];
  for(i=0; i<4; i++)    k[2][i+4] ^= hconst[gindex[index][1]][i];
  mdsh_mul(&k[2][0], &k[2][0], 0xb);  mdsh_mul(&k[2][4], &k[2][4], 0x3);
  mdsh_mul(&k[3][0], &k[3][0], 0xb);  mdsh_mul(&k[3][4], &k[3][4], 0x3);
  for(i=0; i<4; i++) {kout[2][i] = fout[i]^k[2][i]; kout[3][i] = fout[i+4]^k[2][i+4];}
  for(i=0; i<4; i++) {kout[4][i] = fout[i]^k[3][i]; kout[5][i] = fout[i+4]^k[3][i+4];}
  for(i=0; i<4; i++) {kout[6][i] = k[i][i]^k[3][i]; kout[7][i] = k[i][i+4]^k[3][i+4];}
  for(i=0; i<4; i++) {k[3][i] ^= k[2][i+4]; k[3][i+4] ^= k[2][i];  } /* {P(32)}^-1 */
  for(i=0; i<4; i++) {k[2][i] ^= k[3][i]; k[2][i+4] ^= k[3][i+4];  } /* {P(32)}^-1 */
}
void hcrypt_setkey(BYTE *key, BYTE ks[MAX_ROUND+1][8][4], int key_len)
{
  BYTE  k[4][8], fout[8];
  int   i, j, pos, r;
  /* copy & padding */
  if (key_len == 128) {
    for(pos=i=0; i<2; i++)
      for(j=0; j<8; j++)        k[i][j] = key[pos++];
    for(j=0; j<8; j++)    k[2][j] = key[j];
    for(i=0; i<4; i++)    k[3][i] = hconst[3][i];
    for(i=0; i<4; i++)    k[3][i+4] = hconst[2][i];
  }
  else if (key_len == 192) {
    for(pos=i=0; i<3; i++)
      for(j=0; j<8; j++)        k[i][j] = key[pos++];
    for(i=0; i<4; i++)    k[3][i] = hconst[2][i];
    for(i=0; i<4; i++)    k[3][i+4] = hconst[3][i];
  }
  else {
    for(pos=i=0; i<4; i++)
      for(j=0; j<8; j++)        k[i][j] = key[pos++];
  }
  mdsh_mul(&k[2][0], &k[2][0], 5);  mdsh_mul(&k[2][4], &k[2][4], 0xe);
  mdsh_mul(&k[3][0], &k[3][0], 5);  mdsh_mul(&k[3][4], &k[3][4], 0xe);
  for(i=0; i<4; i++)    k[2][i] ^= hconst[gindex[5][0]][i];
  for(i=0; i<4; i++)    k[2][i+4] ^= hconst[gindex[5][1]][i];
  hcrypt_keyf(&k[1][0], fout, &k[2][0]);
  for(i=0; i<8; i++)    k[0][i] ^= fout[i];     /* L+f(R) */
  swap_key(&k[0][0], &k[1][0]);
  if (key_len == 128) {
    for(r=0; r<4; r++)    hcrypt_keyp(&k[0], &ks[r][0], kconst[0][r]);
    for(r=4; r<7; r++)    hcrypt_keyc(&k[0], &ks[r][0], kconst[0][r]);  }
  else if (key_len == 192) {
    for(r=0; r<4; r++)    hcrypt_keyp(&k[0], &ks[r][0], kconst[1][r]);
    for(r=4; r<8; r++)    hcrypt_keyc(&k[0], &ks[r][0], kconst[1][r]);  }
  else {
    for(r=0; r<5; r++)    hcrypt_keyp(&k[0], &ks[r][0], kconst[2][r]);
    for(r=5; r<9; r++)    hcrypt_keyc(&k[0], &ks[r][0], kconst[2][r]);  }
}
void hcrypt_key_edtrans(BYTE eks[MAX_ROUND+1][8][4], BYTE dks[MAX_ROUND+1][8][4], int key_len)
{
  int   i, j, k, r;
  r = key_len/64+4;
  for(j=0; j<4; j++)
    for(k=0; k<4; k++)    dks[0][j][k] = eks[r][j][k];
  for(i=1; i<r; i++) {
    for(j=4; j<8; j++)    hcrypt_imds1(&eks[r-i][j][0], &dks[i-1][j][0]);
    hcrypt_imdsh(&eks[r-i][0], &dks[i][0]);
  }
  for(j=4; j<8; j++)    hcrypt_imds1(&eks[0][j][0], &dks[r-1][j][0]);
  for(j=0; j<4; j++)
    for(k=0; k<4; k++)    dks[r][j][k] = eks[0][j][k];
}
int main()
{
  BYTE key[32], in[16], out[16], res[16];
  BYTE  ks[MAX_ROUND+1][8][4], ks2[MAX_ROUND+1][8][4];
  hcrypt_setkey(key, ks, keylen);           /* 拡大鍵生成 */
  hcrypt_encrypt(in, out, ks, keylen);  /* 暗号化 */
  hcrypt_key_edtrans(ks, ks2, keylen);  /* 復号用鍵生成 (ks->ks2 変換) */
  hcrypt_decrypt(out, res, ks2, keylen);    /* 復号 */
  return EXIT_SUCCESS;
}
```

テストデータ

```
Hierocrypt-L1 (key length = 128 bits)
  key:        4703c87e 817842c4 ce6b167d 43701b76
  plaintext:  85693846 db4c1b34
  ciphertext: 0cb19444 abd24347
Hierocrypt-3 (key length = 128 bits)
  key:        4703c87e 817842c4 ce6b167d 43701b76
  plaintext:  85693846 db4c1b34 87272e55 5761c7f5
  ciphertext: 5c5f4b00 aec36d89 3cf1041e 7fa8bae8
Hierocrypt-3 (key length = 192 bits)
  key:        7742a038 89b58601 f74d5513 88872377 324fbc1d 30c54fc6
  plaintext:  54406620 9d931b33 0c9089fd b4cb8259
  ciphertext: 1d310598 8dbbd50c b0a17193 ea5ba244
Hierocrypt-3 (key length = 256 bits)
  key:        11a18026 9a78dda4 99474621 3b5a6dd6 e34ffe0c c465d593 aff66e13 29419c94
  plaintext:  c16d7efc a1cbafc7 625cbe9c 2593de2d
  ciphertext: c86cd3b4 a3185232 e3467d63 8c6515c9
                                                              (in hexadecimal)
```