Serpent　を利用した暗号ソフト

（C）宇山　靖政

2012.04.08

Serpent 暗号　について

DES が制定されたのは 1977 年でありその後の暗号解読技術の発展、コンピュータの高性能化から、米国商務省標準技術局(NIST)によって新しい暗号技術の選定作業が行われました。
米国政府の次世代標準暗号化方式を決めることになり、NIST は DES に代わる次世代の暗号標準として、AES 候補となる暗号方式を全世界から公募しました。世界中から集まった 15 の方式が審査を受けていたが、１位は、2000 年 10 月に、Joan Daemen 氏と Vincent Rijmen 氏が開発した「Rijndael」でした。
２位がこの　Serpent 暗号　でした。
特徴は、古くから使われている安定した技術を基本にして新しい暗号を提供している所にあります。
サンプルコードの無償提供が行われています。

ソースコードの　MODE_CFB1 の部分は誤りがあると考えて修正しました。
修正には、Twofish 暗号の方法を利用しました。
作者からの返事はまだもらっていませんので、
修正が正しいか否かは自分で元のソースコードと私の修正案を比較して判断してください。

比較しやすいように、3種類のソースコードを掲載いたします。
1．本来の、Serpent のソースコード
2．SerpentEC のソースコード
3．SerpentDC のソースコード
このうち、2, 3についての二次著作権の主張はいたしません。ご自由に変更して下さい。ただし、自作のソフトの中にソースコードを組み込む場合や、鍵の長さが56ビットを超える場合には、貿易管理令についてもご確認下さい。

1．本来の、Serpent のソースコード

serpensboxes.h

```
/*****************************************************************************/
/* Copyright (C) 1998 Ross Anderson, Eli Biham, Lars Knudsen
 * All rights reserved.
 *
 * This code is freely distributed for AES selection process.
 * No other use is allowed.
 *
 * Copyright remains of the copyright holders, and as such any Copyright
 * notices in the code are not to be removed.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted only for the AES selection process, provided
 * that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
```

```
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHORS OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 * The licence and distribution terms for any publically available version or
 * derivative of this code cannot be changed without the authors permission.
 *  i.e. this code cannot simply be copied and put under another distribution
 * licence [including the GNU Public Licence.]
 */

/* S0:  3  8 15  1 10  6  5 11 14 13  4  2  7  0  9 12 */

/* depth = 5,7,4,2, Total gates=18 */
#define RND00(a,b,c,d,w,x,y,z) ¥
        { register unsigned long t02, t03, t05, t06, t07, t08, t09, t11, t12, t13, t14, t15, t17, t01;¥
        t01 = b   ^ c  ; ¥
        t02 = a   | d  ; ¥
        t03 = a   ^ b  ; ¥
        z   = t02 ^ t01; ¥
        t05 = c   | z  ; ¥
        t06 = a   ^ d  ; ¥
        t07 = b   | c  ; ¥
        t08 = d   & t05; ¥
        t09 = t03 & t07; ¥
        y   = t09 ^ t08; ¥
        t11 = t09 & y  ; ¥
        t12 = c   ^ d  ; ¥
        t13 = t07 ^ t11; ¥
        t14 = b   & t06; ¥
        t15 = t06 ^ t13; ¥
        w   =     ~ t15; ¥
        t17 = w   ^ t14; ¥
        x   = t12 ^ t17; }

/* InvS0:  13  3 11  0 10  6  5 12  1 14  4  7 15  9  8  2 */

/* depth = 8,4,3,6, Total gates=19 */
#define InvRND00(a,b,c,d,w,x,y,z) ¥
        { register unsigned long t02, t03, t04, t05, t06, t08, t09, t10, t12, t13, t14, t15, t17, t18,
t01;¥
        t01 = c   ^ d  ; ¥
        t02 = a   | b  ; ¥
```

```
        t03 = b   | c  ; ¥
        t04 = c   & t01; ¥
        t05 = t02 ^ t01; ¥
        t06 = a   | t04; ¥
        y   =     ~ t05; ¥
        t08 = b   ^ d  ; ¥
        t09 = t03 & t08; ¥
        t10 = d   | y  ; ¥
        x   = t09 ^ t06; ¥
        t12 = a   | t05; ¥
        t13 = x   ^ t12; ¥
        t14 = t03 ^ t10; ¥
        t15 = a   ^ c  ; ¥
        z   = t14 ^ t13; ¥
        t17 = t05 & t13; ¥
        t18 = t14 | t17; ¥
        w   = t15 ^ t18; }

/* S1:  15 12  2  7  9  0  5 10  1 11 14  8  6 13  3  4 */

/* depth = 10,7,3,5, Total gates=18 */
#define RND01(a,b,c,d,w,x,y,z) ¥
        { register unsigned long t02, t03, t04, t05, t06, t07, t08, t10, t11, t12, t13, t16, t17, t01;¥
        t01 = a   | d  ; ¥
        t02 = c   ^ d  ; ¥
        t03 =     ~ b  ; ¥
        t04 = a   ^ c  ; ¥
        t05 = a   | t03; ¥
        t06 = d   & t04; ¥
        t07 = t01 & t02; ¥
        t08 = b   | t06; ¥
        y   = t02 ^ t05; ¥
        t10 = t07 ^ t08; ¥
        t11 = t01 ^ t10; ¥
        t12 = y   ^ t11; ¥
        t13 = b   & d  ; ¥
        z   =     ~ t10; ¥
        x   = t13 ^ t12; ¥
        t16 = t10 | x  ; ¥
        t17 = t05 & t16; ¥
        w   = c   ^ t17; }

/* InvS1:   5  8  2 14 15  6 12  3 11  4  7  9  1 13 10  0 */

/* depth = 7,4,5,3, Total gates=18 */
#define InvRND01(a,b,c,d,w,x,y,z) ¥
        { register unsigned long t02, t03, t04, t05, t06, t07, t08, t09, t10, t11, t14, t15, t17, t01;¥
        t01 = a   ^ b  ; ¥
        t02 = b   | d  ; ¥
        t03 = a   & c  ; ¥
        t04 = c   ^ t02; ¥
        t05 = a   | t04; ¥
```

```
        t06 = t01 & t05; ¥
        t07 = d   | t03; ¥
        t08 = b   ^ t06; ¥
        t09 = t07 ^ t06; ¥
        t10 = t04 | t03; ¥
        t11 = d   & t08; ¥
        y   =     ~ t09; ¥
        x   = t10 ^ t11; ¥
        t14 = a   | y  ; ¥
        t15 = t06 ^ x  ; ¥
        z   = t01 ^ t04; ¥
        t17 = c   ^ t15; ¥
        w   = t14 ^ t17; }

/* S2:   8  6  7  9  3 12 10 15 13  1 14  4  0 11  5  2 */

/* depth = 3,8,11,7, Total gates=16 */
#define RND02(a,b,c,d,w,x,y,z) ¥
        { register unsigned long t02, t03, t05, t06, t07, t08, t09, t10, t12, t13, t14, t01;¥
        t01 = a   | c  ; ¥
        t02 = a   ^ b  ; ¥
        t03 = d   ^ t01; ¥
        w   = t02 ^ t03; ¥
        t05 = c   ^ w  ; ¥
        t06 = b   ^ t05; ¥
        t07 = b   | t05; ¥
        t08 = t01 & t06; ¥
        t09 = t03 ^ t07; ¥
        t10 = t02 | t09; ¥
        x   = t10 ^ t08; ¥
        t12 = a   | d  ; ¥
        t13 = t09 ^ x  ; ¥
        t14 = b   ^ t13; ¥
        z   =     ~ t09; ¥
        y   = t12 ^ t14; }

/* InvS2:  12  9 15  4 11 14  1  2  0  3  6 13  5  8 10  7 */

/* depth = 3,6,8,3, Total gates=18 */
#define InvRND02(a,b,c,d,w,x,y,z) ¥
        { register unsigned long t02, t03, t04, t06, t07, t08, t09, t10, t11, t12, t15, t16, t17, t01;¥
        t01 = a   ^ d  ; ¥
        t02 = c   ^ d  ; ¥
        t03 = a   & c  ; ¥
        t04 = b   | t02; ¥
        w   = t01 ^ t04; ¥
        t06 = a   | c  ; ¥
        t07 = d   | w  ; ¥
        t08 =     ~ d  ; ¥
        t09 = b   & t06; ¥
        t10 = t08 | t03; ¥
        t11 = b   & t07; ¥
```

```
        t12 = t06 & t02; ¥
        z   = t09 ^ t10; ¥
        x   = t12 ^ t11; ¥
        t15 = c   & z  ; ¥
        t16 = w   ^ x  ; ¥
        t17 = t10 ^ t15; ¥
        y   = t16 ^ t17; }

/* S3:   0 15 11  8 12  9  6  3 13  1  2  4 10  7  5 14 */

/* depth = 8,3,5,5, Total gates=18 */
#define RND03(a,b,c,d,w,x,y,z) ¥
        { register unsigned long t02, t03, t04, t05, t06, t07, t08, t09, t10, t11, t13, t14, t15, t01;¥
        t01 = a   ^ c  ; ¥
        t02 = a   | d  ; ¥
        t03 = a   & d  ; ¥
        t04 = t01 & t02; ¥
        t05 = b   | t03; ¥
        t06 = a   & b  ; ¥
        t07 = d   ^ t04; ¥
        t08 = c   | t06; ¥
        t09 = b   ^ t07; ¥
        t10 = d   & t05; ¥
        t11 = t02 ^ t10; ¥
        z   = t08 ^ t09; ¥
        t13 = d   | z  ; ¥
        t14 = a   | t07; ¥
        t15 = b   & t13; ¥
        y   = t08 ^ t11; ¥
        w   = t14 ^ t15; ¥
        x   = t05 ^ t04; }

/* InvS3:   0  9 10  7 11 14  6 13  3  5 12  2  4  8 15  1 */

/* depth = 3,6,4,4, Total gates=17 */
#define InvRND03(a,b,c,d,w,x,y,z) ¥
        { register unsigned long t02, t03, t04, t05, t06, t07, t09, t11, t12, t13, t14, t16, t01;¥
        t01 = c   | d  ; ¥
        t02 = a   | d  ; ¥
        t03 = c   ^ t02; ¥
        t04 = b   ^ t02; ¥
        t05 = a   ^ d  ; ¥
        t06 = t04 & t03; ¥
        t07 = b   & t01; ¥
        y   = t05 ^ t06; ¥
        t09 = a   ^ t03; ¥
        w   = t07 ^ t03; ¥
        t11 = w   | t05; ¥
        t12 = t09 & t11; ¥
        t13 = a   & y  ; ¥
        t14 = t01 ^ t05; ¥
        x   = b   ^ t12; ¥
```

```
        t16 = b   | t13; ¥
        z   = t14 ^ t16; }

/* S4:   1 15  8  3 12  0 11  6  2  5  4 10  9 14  7 13 */

/* depth = 6,7,5,3, Total gates=19 */
#define RND04(a,b,c,d,w,x,y,z) ¥
        { register unsigned long t02, t03, t04, t05, t06, t08, t09, t10, t11, t12, t13, t14, t15, t16,
t01;¥
        t01 = a   | b  ; ¥
        t02 = b   | c  ; ¥
        t03 = a   ^ t02; ¥
        t04 = b   ^ d  ; ¥
        t05 = d   | t03; ¥
        t06 = d   & t01; ¥
        z   = t03 ^ t06; ¥
        t08 = z   & t04; ¥
        t09 = t04 & t05; ¥
        t10 = c   ^ t06; ¥
        t11 = b   & c  ; ¥
        t12 = t04 ^ t08; ¥
        t13 = t11 | t03; ¥
        t14 = t10 ^ t09; ¥
        t15 = a   & t05; ¥
        t16 = t11 | t12; ¥
        y   = t13 ^ t08; ¥
        x   = t15 ^ t16; ¥
        w   =     ~ t14; }

/* InvS4:   5  0  8  3 10  9  7 14  2 12 11  6  4 15 13  1 */

/* depth = 6,4,7,3, Total gates=17 */
#define InvRND04(a,b,c,d,w,x,y,z) ¥
        { register unsigned long t02, t03, t04, t05, t06, t07, t09, t10, t11, t12, t13, t15, t01;¥
        t01 = b   | d  ; ¥
        t02 = c   | d  ; ¥
        t03 = a   & t01; ¥
        t04 = b   ^ t02; ¥
        t05 = c   ^ d  ; ¥
        t06 =     ~ t03; ¥
        t07 = a   & t04; ¥
        x   = t05 ^ t07; ¥
        t09 = x   | t06; ¥
        t10 = a   ^ t07; ¥
        t11 = t01 ^ t09; ¥
        t12 = d   ^ t04; ¥
        t13 = c   | t10; ¥
        z   = t03 ^ t12; ¥
        t15 = a   ^ t04; ¥
        y   = t11 ^ t13; ¥
        w   = t15 ^ t09; }
```

```
/* S5:  15  5  2 11  4 10  9 12  0  3 14  8 13  6  7  1 */

/* depth = 4,6,8,6, Total gates=17 */
#define RND05(a,b,c,d,w,x,y,z) ¥
        { register unsigned long t02, t03, t04, t05, t07, t08, t09, t10, t11, t12, t13, t14, t01;¥
        t01 = b   ^ d  ; ¥
        t02 = b   | d  ; ¥
        t03 = a   & t01; ¥
        t04 = c   ^ t02; ¥
        t05 = t03 ^ t04; ¥
        w   =     ~ t05; ¥
        t07 = a   ^ t01; ¥
        t08 = d   | w  ; ¥
        t09 = b   | t05; ¥
        t10 = d   ^ t08; ¥
        t11 = b   | t07; ¥
        t12 = t03 | w  ; ¥
        t13 = t07 | t10; ¥
        t14 = t01 ^ t11; ¥
        y   = t09 ^ t13; ¥
        x   = t07 ^ t08; ¥
        z   = t12 ^ t14; }

/* InvS5:   8 15  2  9  4  1 13 14 11  6  5  3  7 12 10  0 */

/* depth = 4,6,9,7, Total gates=17 */
#define InvRND05(a,b,c,d,w,x,y,z) ¥
        { register unsigned long t02, t03, t04, t05, t07, t08, t09, t10, t12, t13, t15, t16, t01;¥
        t01 = a   & d  ; ¥
        t02 = c   ^ t01; ¥
        t03 = a   ^ d  ; ¥
        t04 = b   & t02; ¥
        t05 = a   & c  ; ¥
        w   = t03 ^ t04; ¥
        t07 = a   & w  ; ¥
        t08 = t01 ^ w  ; ¥
        t09 = b   | t05; ¥
        t10 =     ~ b  ; ¥
        x   = t08 ^ t09; ¥
        t12 = t10 | t07; ¥
        t13 = w   | x  ; ¥
        z   = t02 ^ t12; ¥
        t15 = t02 ^ t13; ¥
        t16 = b   ^ d  ; ¥
        y   = t16 ^ t15; }

/* S6:   7  2 12  5  8  4  6 11 14  9  1 15 13  3 10  0 */

/* depth = 8,3,6,3, Total gates=19 */
#define RND06(a,b,c,d,w,x,y,z) ¥
        { register unsigned long t02, t03, t04, t05, t07, t08, t09, t10, t11, t12, t13, t15, t17, t18,
t01;¥
```

```
        t01 = a   & d  ; ¥
        t02 = b   ^ c  ; ¥
        t03 = a   ^ d  ; ¥
        t04 = t01 ^ t02; ¥
        t05 = b   | c  ; ¥
        x   =     ~ t04; ¥
        t07 = t03 & t05; ¥
        t08 = b   & x  ; ¥
        t09 = a   | c  ; ¥
        t10 = t07 ^ t08; ¥
        t11 = b   | d  ; ¥
        t12 = c   ^ t11; ¥
        t13 = t09 ^ t10; ¥
        y   =     ~ t13; ¥
        t15 = x   & t03; ¥
        z   = t12 ^ t07; ¥
        t17 = a   ^ b  ; ¥
        t18 = y   ^ t15; ¥
        w   = t17 ^ t18; }

/* InvS6:  15 10  1 13  5  3  6  0  4  9 14  7  2 12  8 11 */

/* depth = 5,3,8,6, Total gates=19 */
#define InvRND06(a,b,c,d,w,x,y,z) ¥
        { register unsigned long t02, t03, t04, t05, t06, t07, t08, t09, t12, t13, t14, t15, t16, t17,
t01;¥
        t01 = a   ^ c  ; ¥
        t02 =     ~ c  ; ¥
        t03 = b   & t01; ¥
        t04 = b   | t02; ¥
        t05 = d   | t03; ¥
        t06 = b   ^ d  ; ¥
        t07 = a   & t04; ¥
        t08 = a   | t02; ¥
        t09 = t07 ^ t05; ¥
        x   = t06 ^ t08; ¥
        w   =     ~ t09; ¥
        t12 = b   & w  ; ¥
        t13 = t01 & t05; ¥
        t14 = t01 ^ t12; ¥
        t15 = t07 ^ t13; ¥
        t16 = d   | t02; ¥
        t17 = a   ^ x  ; ¥
        z   = t17 ^ t15; ¥
        y   = t16 ^ t14; }

/* S7:   1 13 15  0 14  8  2 11  7  4 12 10  9  3  5  6 */

/* depth = 10,7,10,4, Total gates=19 */
#define RND07(a,b,c,d,w,x,y,z) ¥
        { register unsigned long t02, t03, t04, t05, t06, t08, t09, t10, t11, t13, t14, t15, t16, t17,
t01;¥
```

```
        t01 = a   & c  ; ¥
        t02 =     ~ d  ; ¥
        t03 = a   & t02; ¥
        t04 = b   | t01; ¥
        t05 = a   & b  ; ¥
        t06 = c   ^ t04; ¥
        z   = t03 ^ t06; ¥
        t08 = c   | z  ; ¥
        t09 = d   | t05; ¥
        t10 = a   ^ t08; ¥
        t11 = t04 & z  ; ¥
        x   = t09 ^ t10; ¥
        t13 = b   ^ x  ; ¥
        t14 = t01 ^ x  ; ¥
        t15 = c   ^ t05; ¥
        t16 = t11 | t13; ¥
        t17 = t02 | t14; ¥
        w   = t15 ^ t17; ¥
        y   = a   ^ t16; }

/* InvS7:   3  0  6 13  9 14 15  8  5 12 11  7 10  1  4  2 */

/* depth = 9,7,3,3, Total gates=18 */
#define InvRND07(a,b,c,d,w,x,y,z) ¥
        { register unsigned long t02, t03, t04, t06, t07, t08, t09, t10, t11, t13, t14, t15, t16, t01;¥
        t01 = a   & b  ; ¥
        t02 = a   | b  ; ¥
        t03 = c   | t01; ¥
        t04 = d   & t02; ¥
        z   = t03 ^ t04; ¥
        t06 = b   ^ t04; ¥
        t07 = d   ^ z  ; ¥
        t08 =     ~ t07; ¥
        t09 = t06 | t08; ¥
        t10 = b   ^ d  ; ¥
        t11 = a   | d  ; ¥
        x   = a   ^ t09; ¥
        t13 = c   ^ t06; ¥
        t14 = c   & t11; ¥
        t15 = d   | x  ; ¥
        t16 = t01 | t10; ¥
        w   = t13 ^ t15; ¥
        y   = t14 ^ t16; }

#define RND08(a,b,c,d,e,f,g,h) RND00(a,b,c,d,e,f,g,h)
#define RND09(a,b,c,d,e,f,g,h) RND01(a,b,c,d,e,f,g,h)
#define RND10(a,b,c,d,e,f,g,h) RND02(a,b,c,d,e,f,g,h)
#define RND11(a,b,c,d,e,f,g,h) RND03(a,b,c,d,e,f,g,h)
#define RND12(a,b,c,d,e,f,g,h) RND04(a,b,c,d,e,f,g,h)
#define RND13(a,b,c,d,e,f,g,h) RND05(a,b,c,d,e,f,g,h)
#define RND14(a,b,c,d,e,f,g,h) RND06(a,b,c,d,e,f,g,h)
#define RND15(a,b,c,d,e,f,g,h) RND07(a,b,c,d,e,f,g,h)
```

```
#define RND16(a,b,c,d,e,f,g,h) RND00(a,b,c,d,e,f,g,h)
#define RND17(a,b,c,d,e,f,g,h) RND01(a,b,c,d,e,f,g,h)
#define RND18(a,b,c,d,e,f,g,h) RND02(a,b,c,d,e,f,g,h)
#define RND19(a,b,c,d,e,f,g,h) RND03(a,b,c,d,e,f,g,h)
#define RND20(a,b,c,d,e,f,g,h) RND04(a,b,c,d,e,f,g,h)
#define RND21(a,b,c,d,e,f,g,h) RND05(a,b,c,d,e,f,g,h)
#define RND22(a,b,c,d,e,f,g,h) RND06(a,b,c,d,e,f,g,h)
#define RND23(a,b,c,d,e,f,g,h) RND07(a,b,c,d,e,f,g,h)
#define RND24(a,b,c,d,e,f,g,h) RND00(a,b,c,d,e,f,g,h)
#define RND25(a,b,c,d,e,f,g,h) RND01(a,b,c,d,e,f,g,h)
#define RND26(a,b,c,d,e,f,g,h) RND02(a,b,c,d,e,f,g,h)
#define RND27(a,b,c,d,e,f,g,h) RND03(a,b,c,d,e,f,g,h)
#define RND28(a,b,c,d,e,f,g,h) RND04(a,b,c,d,e,f,g,h)
#define RND29(a,b,c,d,e,f,g,h) RND05(a,b,c,d,e,f,g,h)
#define RND30(a,b,c,d,e,f,g,h) RND06(a,b,c,d,e,f,g,h)
#define RND31(a,b,c,d,e,f,g,h) RND07(a,b,c,d,e,f,g,h)

#define InvRND08(a,b,c,d,e,f,g,h) InvRND00(a,b,c,d,e,f,g,h)
#define InvRND09(a,b,c,d,e,f,g,h) InvRND01(a,b,c,d,e,f,g,h)
#define InvRND10(a,b,c,d,e,f,g,h) InvRND02(a,b,c,d,e,f,g,h)
#define InvRND11(a,b,c,d,e,f,g,h) InvRND03(a,b,c,d,e,f,g,h)
#define InvRND12(a,b,c,d,e,f,g,h) InvRND04(a,b,c,d,e,f,g,h)
#define InvRND13(a,b,c,d,e,f,g,h) InvRND05(a,b,c,d,e,f,g,h)
#define InvRND14(a,b,c,d,e,f,g,h) InvRND06(a,b,c,d,e,f,g,h)
#define InvRND15(a,b,c,d,e,f,g,h) InvRND07(a,b,c,d,e,f,g,h)
#define InvRND16(a,b,c,d,e,f,g,h) InvRND00(a,b,c,d,e,f,g,h)
#define InvRND17(a,b,c,d,e,f,g,h) InvRND01(a,b,c,d,e,f,g,h)
#define InvRND18(a,b,c,d,e,f,g,h) InvRND02(a,b,c,d,e,f,g,h)
#define InvRND19(a,b,c,d,e,f,g,h) InvRND03(a,b,c,d,e,f,g,h)
#define InvRND20(a,b,c,d,e,f,g,h) InvRND04(a,b,c,d,e,f,g,h)
#define InvRND21(a,b,c,d,e,f,g,h) InvRND05(a,b,c,d,e,f,g,h)
#define InvRND22(a,b,c,d,e,f,g,h) InvRND06(a,b,c,d,e,f,g,h)
#define InvRND23(a,b,c,d,e,f,g,h) InvRND07(a,b,c,d,e,f,g,h)
#define InvRND24(a,b,c,d,e,f,g,h) InvRND00(a,b,c,d,e,f,g,h)
#define InvRND25(a,b,c,d,e,f,g,h) InvRND01(a,b,c,d,e,f,g,h)
#define InvRND26(a,b,c,d,e,f,g,h) InvRND02(a,b,c,d,e,f,g,h)
#define InvRND27(a,b,c,d,e,f,g,h) InvRND03(a,b,c,d,e,f,g,h)
#define InvRND28(a,b,c,d,e,f,g,h) InvRND04(a,b,c,d,e,f,g,h)
#define InvRND29(a,b,c,d,e,f,g,h) InvRND05(a,b,c,d,e,f,g,h)
#define InvRND30(a,b,c,d,e,f,g,h) InvRND06(a,b,c,d,e,f,g,h)
#define InvRND31(a,b,c,d,e,f,g,h) InvRND07(a,b,c,d,e,f,g,h)

/* Linear transformations and key mixing: */

#define ROL(x,n) ((((unsigned long)(x))<<(n))| ¥
                  (((unsigned long)(x))>>(32-(n))))
#define ROR(x,n) ((((unsigned long)(x))<<(32-(n)))| ¥
                  (((unsigned long)(x))>>(n)))

#define transform(x0, x1, x2, x3, y0, y1, y2, y3) ¥
      y0 = ROL(x0, 13); ¥
      y2 = ROL(x2, 3); ¥
```

```
	y1 = x1 ^ y0 ^ y2; ¥
	y3 = x3 ^ y2 ^ ((unsigned long)y0)<<3; ¥
	y1 = ROL(y1, 1); ¥
	y3 = ROL(y3, 7); ¥
	y0 = y0 ^ y1 ^ y3; ¥
	y2 = y2 ^ y3 ^ ((unsigned long)y1<<7); ¥
	y0 = ROL(y0, 5); ¥
	y2 = ROL(y2, 22)

#define inv_transform(x0, x1, x2, x3, y0, y1, y2, y3) ¥
	y2 = ROR(x2, 22);¥
	y0 = ROR(x0, 5); ¥
	y2 = y2 ^ x3 ^ ((unsigned long)x1<<7); ¥
	y0 = y0 ^ x1 ^ x3; ¥
	y3 = ROR(x3, 7); ¥
	y1 = ROR(x1, 1); ¥
	y3 = y3 ^ y2 ^ ((unsigned long)y0)<<3; ¥
	y1 = y1 ^ y0 ^ y2; ¥
	y2 = ROR(y2, 3); ¥
	y0 = ROR(y0, 13)

#define keying(x0, x1, x2, x3, subkey) ¥
			x0^=subkey[0];x1^=subkey[1]; ¥
			x2^=subkey[2];x3^=subkey[3]

/* PHI: Constant used in the key schedule */
#define PHI 0x9e3779b9L
```

Serpent.h

```
/*************************************************************/
/*  aes.h  */
/*************************************************************/

/*  AES Cipher header file for ANSI C Submissions
      Lawrence E. Bassham III
      Computer Security Division
      National Institute of Standards and Technology

      April 15, 1998

    This sample is to assist implementers developing to the Cryptographic
API Profile for AES Candidate Algorithm Submissions.  Please consult this
document as a cross-reference.

    ANY CHANGES, WHERE APPROPRIATE, TO INFORMATION PROVIDED IN THIS FILE
MUST BE DOCUMENTED.  CHANGES ARE ONLY APPROPRIATE WHERE SPECIFIED WITH
THE STRING "CHANGE POSSIBLE".  FUNCTION CALLS AND THEIR PARAMETERS CANNOT
BE CHANGED.  STRUCTURES CAN BE ALTERED TO ALLOW IMPLEMENTERS TO INCLUDE
IMPLEMENTATION SPECIFIC INFORMATION.
```

```
*/

/*  Includes:
        Standard include files
*/

#include <stdio.h>

/*  Defines:
        Add any additional defines you need
*/

#define     DIR_ENCRYPT     0    /*  Are we encrpyting?  */
#define     DIR_DECRYPT     1    /*  Are we decrpyting?  */
#define     MODE_ECB        1    /*  Are we ciphering in ECB mode?   */
#define     MODE_CBC        2    /*  Are we ciphering in CBC mode?   */
#define     MODE_CFB1       3    /*  Are we ciphering in 1-bit CFB mode? */
#define     TRUE            1
#define     FALSE           0

/*  Error Codes - CHANGE POSSIBLE: inclusion of additional error codes  */
#define     BAD_KEY_DIR        -1  /*  Key direction is invalid, e;g;,
                                       unknown value */
#define     BAD_KEY_MAT        -2  /*  Key material not of correct
                                       length */
#define     BAD_KEY_INSTANCE   -3  /*  Key passed is not valid  */
#define     BAD_CIPHER_MODE    -4  /*  Params struct passed to
                                       cipherInit invalid */
#define     BAD_CIPHER_STATE   -5  /*  Cipher in wrong state (e.g., not
                                       initialized) */

/*  CHANGE POSSIBLE:  inclusion of algorithm specific defines  */
#define     MAX_KEY_SIZE 64  /* # of ASCII char's needed to
                                       represent a key */
#define     MAX_IV_SIZE          32  /* # of ASCII char's needed to
                                       represent an IV  */

/*  Typedefs:

        Typedef'ed data storage elements.  Add any algorithm specific
parameters at the bottom of the structs as appropriate.
*/

typedef    unsigned char    BYTE;

/*  The structure for key information */
typedef struct {
      BYTE  direction;    /*  Key used for encrypting or decrypting? */
      int   keyLen;       /*  Length of the key  */
      char  keyMaterial[MAX_KEY_SIZE+1];  /*  Raw key data in ASCII, e.g.,
                                       what the user types or KAT values)*/
      /*  The following parameters are algorithm dependent, replace or
```

```
                add as necessary  */
      unsigned long key[8];             /* The key in binary */
      unsigned long subkeys[33][4];       /* Serpent subkeys */
      } keyInstance;

/*  The structure for cipher information */
typedef struct {
      BYTE       mode;           /* MODE_ECB, MODE_CBC, or MODE_CFB1 */
      char  IV[MAX_IV_SIZE];      /* A possible Initialization Vector for
                                        ciphering */
      /*  Add any algorithm specific parameters needed here  */
      int   blockSize;            /* Sample: Handles non-128 bit block sizes
                                        (if available) */
      } cipherInstance;



/*  Function protoypes  */
int makeKey(keyInstance *key, BYTE direction, int keyLen,
            char *keyMaterial);

int cipherInit(cipherInstance *cipher, BYTE mode, char *IV);

int blockEncrypt(cipherInstance *cipher, keyInstance *key, BYTE *input,
                 int inputLen, BYTE *outBuffer);

int blockDecrypt(cipherInstance *cipher, keyInstance *key, BYTE *input,
                 int inputLen, BYTE *outBuffer);
```

serpent.c

```
/*********************************************************************/
/* Copyright (C) 1998 Ross Anderson, Eli Biham, Lars Knudsen
 * All rights reserved.
 *
 * This code is freely distributed for AES selection process.
 * No other use is allowed.
 *
 * Copyright remains of the copyright holders, and as such any Copyright
 * notices in the code are not to be removed.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted only for the AES selection process, provided
 * that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
```

```
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHORS OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 * The licence and distribution terms for any publically available version or
 * derivative of this code cannot be changed without the authors permission.
 *  i.e. this code cannot simply be copied and put under another distribution
 * licence [including the GNU Public Licence.]
 */

#include "serpent.h"
#include "serpentsboxes.h"

/*  The functions  */
int makeKey(keyInstance *key, BYTE direction, int keyLen,
                          char *keyMaterial)
{
  unsigned long i,j;
  unsigned long w[132],k[132];
  int rc;

  if(direction != DIR_ENCRYPT &&
     direction != DIR_DECRYPT)
    return BAD_KEY_DIR;

  if(keyLen>256 || keyLen<1)
    return BAD_KEY_MAT;

  key->direction=direction;
  key->keyLen=keyLen;
  strncpy(key->keyMaterial, keyMaterial, MAX_KEY_SIZE+1);

  rc=serpent_convert_from_string(keyLen, keyMaterial, key->key);
  if(rc<=0)
    return BAD_KEY_MAT;

  for(i=0; i<keyLen/32; i++)
    w[i]=key->key[i];
  if(keyLen<256)
    w[i]=(key->key[i]&((1L<<((keyLen&31)))-1))|(1L<<((keyLen&31)));
  for(i++; i<8; i++)
    w[i]=0;
  for(i=8; i<16; i++)
    w[i]=ROL(w[i-8]^w[i-5]^w[i-3]^w[i-1]^PHI^(i-8),11);
```

```
  for(i=0; i<8; i++)
    w[i]=w[i+8];
  for(i=8; i<132; i++)
    w[i]=ROL(w[i-8]^w[i-5]^w[i-3]^w[i-1]^PHI^i,11);

  RND03(w[  0], w[  1], w[  2], w[  3], k[  0], k[  1], k[  2], k[  3]);
  RND02(w[  4], w[  5], w[  6], w[  7], k[  4], k[  5], k[  6], k[  7]);
  RND01(w[  8], w[  9], w[ 10], w[ 11], k[  8], k[  9], k[ 10], k[ 11]);
  RND00(w[ 12], w[ 13], w[ 14], w[ 15], k[ 12], k[ 13], k[ 14], k[ 15]);
  RND31(w[ 16], w[ 17], w[ 18], w[ 19], k[ 16], k[ 17], k[ 18], k[ 19]);
  RND30(w[ 20], w[ 21], w[ 22], w[ 23], k[ 20], k[ 21], k[ 22], k[ 23]);
  RND29(w[ 24], w[ 25], w[ 26], w[ 27], k[ 24], k[ 25], k[ 26], k[ 27]);
  RND28(w[ 28], w[ 29], w[ 30], w[ 31], k[ 28], k[ 29], k[ 30], k[ 31]);
  RND27(w[ 32], w[ 33], w[ 34], w[ 35], k[ 32], k[ 33], k[ 34], k[ 35]);
  RND26(w[ 36], w[ 37], w[ 38], w[ 39], k[ 36], k[ 37], k[ 38], k[ 39]);
  RND25(w[ 40], w[ 41], w[ 42], w[ 43], k[ 40], k[ 41], k[ 42], k[ 43]);
  RND24(w[ 44], w[ 45], w[ 46], w[ 47], k[ 44], k[ 45], k[ 46], k[ 47]);
  RND23(w[ 48], w[ 49], w[ 50], w[ 51], k[ 48], k[ 49], k[ 50], k[ 51]);
  RND22(w[ 52], w[ 53], w[ 54], w[ 55], k[ 52], k[ 53], k[ 54], k[ 55]);
  RND21(w[ 56], w[ 57], w[ 58], w[ 59], k[ 56], k[ 57], k[ 58], k[ 59]);
  RND20(w[ 60], w[ 61], w[ 62], w[ 63], k[ 60], k[ 61], k[ 62], k[ 63]);
  RND19(w[ 64], w[ 65], w[ 66], w[ 67], k[ 64], k[ 65], k[ 66], k[ 67]);
  RND18(w[ 68], w[ 69], w[ 70], w[ 71], k[ 68], k[ 69], k[ 70], k[ 71]);
  RND17(w[ 72], w[ 73], w[ 74], w[ 75], k[ 72], k[ 73], k[ 74], k[ 75]);
  RND16(w[ 76], w[ 77], w[ 78], w[ 79], k[ 76], k[ 77], k[ 78], k[ 79]);
  RND15(w[ 80], w[ 81], w[ 82], w[ 83], k[ 80], k[ 81], k[ 82], k[ 83]);
  RND14(w[ 84], w[ 85], w[ 86], w[ 87], k[ 84], k[ 85], k[ 86], k[ 87]);
  RND13(w[ 88], w[ 89], w[ 90], w[ 91], k[ 88], k[ 89], k[ 90], k[ 91]);
  RND12(w[ 92], w[ 93], w[ 94], w[ 95], k[ 92], k[ 93], k[ 94], k[ 95]);
  RND11(w[ 96], w[ 97], w[ 98], w[ 99], k[ 96], k[ 97], k[ 98], k[ 99]);
  RND10(w[100], w[101], w[102], w[103], k[100], k[101], k[102], k[103]);
  RND09(w[104], w[105], w[106], w[107], k[104], k[105], k[106], k[107]);
  RND08(w[108], w[109], w[110], w[111], k[108], k[109], k[110], k[111]);
  RND07(w[112], w[113], w[114], w[115], k[112], k[113], k[114], k[115]);
  RND06(w[116], w[117], w[118], w[119], k[116], k[117], k[118], k[119]);
  RND05(w[120], w[121], w[122], w[123], k[120], k[121], k[122], k[123]);
  RND04(w[124], w[125], w[126], w[127], k[124], k[125], k[126], k[127]);
  RND03(w[128], w[129], w[130], w[131], k[128], k[129], k[130], k[131]);

  for(i=0; i<=32; i++)
    for(j=0; j<4; j++)
      key->subkeys[i][j] = k[4*i+j];

  return TRUE;
}

int cipherInit(cipherInstance *cipher, BYTE mode, char *IV)
{
  int i;
  int rc;

  if((mode != MODE_ECB) &&
```

```
      (mode != MODE_CBC) &&
      (mode != MODE_CFB1))
    return BAD_CIPHER_MODE;

  cipher->mode = mode;            /* MODE_ECB, MODE_CBC, or MODE_CFB1 */
  cipher->blockSize=128;
  if(mode != MODE_ECB)
    {
      rc=serpent_convert_from_string(cipher->blockSize, IV, cipher->IV);
      if(rc<=0)
        return BAD_CIPHER_STATE;
    }

  return TRUE;
}

int blockEncrypt(cipherInstance *cipher,
                 keyInstance *key,
                 BYTE *input,
                 int inputLen,
                 BYTE *outBuffer)
{
  unsigned long t[4];
  int i, b;

  /*
   * Note about optimization: the code becomes slower of the calls to
   * serpent_encrypt and serpent_decrypt are replaced by inlined code.
   * (tested on Pentium 133MMX)
   */

  switch(cipher->mode)
    {
    case MODE_ECB:
      for(b=0; b<inputLen; b+=128, input+=16, outBuffer+=16)
        serpent_encrypt(input, outBuffer, key->subkeys);
      return inputLen;

    case MODE_CBC:
      t[0] = ((unsigned long*)cipher->IV)[0];
      t[1] = ((unsigned long*)cipher->IV)[1];
      t[2] = ((unsigned long*)cipher->IV)[2];
      t[3] = ((unsigned long*)cipher->IV)[3];
      for(b=0; b<inputLen; b+=128, input+=16, outBuffer+=16)
        {
          t[0] ^= ((unsigned long*)input)[0];
          t[1] ^= ((unsigned long*)input)[1];
          t[2] ^= ((unsigned long*)input)[2];
          t[3] ^= ((unsigned long*)input)[3];
          serpent_encrypt(t, t, key->subkeys);
          ((unsigned long*)outBuffer)[0] = t[0];
          ((unsigned long*)outBuffer)[1] = t[1];
```

```
            ((unsigned long*)outBuffer)[2] = t[2];
            ((unsigned long*)outBuffer)[3] = t[3];
          }
        ((unsigned long*)cipher->IV)[0] = t[0];
        ((unsigned long*)cipher->IV)[1] = t[1];
        ((unsigned long*)cipher->IV)[2] = t[2];
        ((unsigned long*)cipher->IV)[3] = t[3];
        return inputLen;

      case MODE_CFB1:
        t[0] = ((unsigned long*)cipher->IV)[0];
        t[1] = ((unsigned long*)cipher->IV)[1];
        t[2] = ((unsigned long*)cipher->IV)[2];
        t[3] = ((unsigned long*)cipher->IV)[3];
        for(b=0; b<inputLen; b+=8, input++, outBuffer++)
          {
            int bit;
            int bytedata = (input[0])&0xFF;

            for(bit=0; bit<8; bit++)
              {
                unsigned long tt[4];

                serpent_encrypt(t, tt, key->subkeys);

                bytedata ^= (tt[0]&1);

                tt[0] = ((tt[0]>>1)&0x7FFFFFFF) | ((tt[1]&1)<<31);
                tt[1] = ((tt[1]>>1)&0x7FFFFFFF) | ((tt[2]&1)<<31);
                tt[2] = ((tt[2]>>1)&0x7FFFFFFF) | ((tt[3]&1)<<31);
                tt[3] = ((tt[3]>>1)&0x7FFFFFFF) | ((bytedata&1)<<31);

                bytedata = bytedata>>1;
              }
            outBuffer[0] = (t[3]>>24)&0xFF;
          }
        ((unsigned long*)cipher->IV)[0] = t[0];
        ((unsigned long*)cipher->IV)[1] = t[1];
        ((unsigned long*)cipher->IV)[2] = t[2];
        ((unsigned long*)cipher->IV)[3] = t[3];
        return inputLen;

      default:
        return BAD_CIPHER_STATE;
      }
  }

  int blockDecrypt(cipherInstance *cipher,
                   keyInstance *key,
                   BYTE *input,
                   int inputLen,
                   BYTE *outBuffer)
```

```
{
  unsigned long t[4];
  int i, b;

  switch(cipher->mode)
    {
    case MODE_ECB:
      for(b=0; b<inputLen; b+=128, input+=16, outBuffer+=16)
        serpent_decrypt(input, outBuffer, key->subkeys);
      return inputLen;

    case MODE_CBC:
      t[0] = ((unsigned long*)cipher->IV)[0];
      t[1] = ((unsigned long*)cipher->IV)[1];
      t[2] = ((unsigned long*)cipher->IV)[2];
      t[3] = ((unsigned long*)cipher->IV)[3];
      for(b=0; b<inputLen; b+=128, input+=16, outBuffer+=16)
        {
          serpent_decrypt(input, outBuffer, key->subkeys);
          ((unsigned long*)outBuffer)[0] ^= t[0];
          ((unsigned long*)outBuffer)[1] ^= t[1];
          ((unsigned long*)outBuffer)[2] ^= t[2];
          ((unsigned long*)outBuffer)[3] ^= t[3];
          t[0] = ((unsigned long*)input)[0];
          t[1] = ((unsigned long*)input)[1];
          t[2] = ((unsigned long*)input)[2];
          t[3] = ((unsigned long*)input)[3];
        }
      ((unsigned long*)cipher->IV)[0] = t[0];
      ((unsigned long*)cipher->IV)[1] = t[1];
      ((unsigned long*)cipher->IV)[2] = t[2];
      ((unsigned long*)cipher->IV)[3] = t[3];
      return inputLen;

    case MODE_CFB1:
      t[0] = ((unsigned long*)cipher->IV)[0];
      t[1] = ((unsigned long*)cipher->IV)[1];
      t[2] = ((unsigned long*)cipher->IV)[2];
      t[3] = ((unsigned long*)cipher->IV)[3];
      for(b=0; b<inputLen; b+=8, input++, outBuffer++)
        {
          int bit;
          int bytedata = (input[0])&0xFF;
          int outdata=0;

          for(bit=0; bit<8; bit++)
            {
              unsigned long tt[4];

              serpent_encrypt(t, tt, key->subkeys);

              outdata |= ((bytedata^tt[0])&1)<<bit;
```

```
              tt[0] = ((tt[0]>>1)&0x7FFFFFFF) | ((tt[1]&1)<<31);
              tt[1] = ((tt[1]>>1)&0x7FFFFFFF) | ((tt[2]&1)<<31);
              tt[2] = ((tt[2]>>1)&0x7FFFFFFF) | ((tt[3]&1)<<31);
              tt[3] = ((tt[3]>>1)&0x7FFFFFFF) | ((bytedata&1)<<31);

              bytedata = bytedata>>1;
            }
          outBuffer[0] = outdata;
        }
      ((unsigned long*)cipher->IV)[0] = t[0];
      ((unsigned long*)cipher->IV)[1] = t[1];
      ((unsigned long*)cipher->IV)[2] = t[2];
      ((unsigned long*)cipher->IV)[3] = t[3];
      return inputLen;

    default:
      return BAD_CIPHER_STATE;
    }
}

serpent_encrypt(unsigned long plaintext[4],
                unsigned long ciphertext[4],
                unsigned long subkeys[33][4])
{
  register unsigned long x0, x1, x2, x3;
  register unsigned long y0, y1, y2, y3;

  x0=plaintext[0];
  x1=plaintext[1];
  x2=plaintext[2];
  x3=plaintext[3];

  /* Start to encrypt the plaintext x */
  keying(x0, x1, x2, x3, subkeys[ 0]);
  RND00(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[ 1]);
  RND01(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[ 2]);
  RND02(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[ 3]);
  RND03(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[ 4]);
  RND04(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[ 5]);
  RND05(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
```

```
keying(x0, x1, x2, x3, subkeys[ 6]);
RND06(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[ 7]);
RND07(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[ 8]);
RND08(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[ 9]);
RND09(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[10]);
RND10(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[11]);
RND11(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[12]);
RND12(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[13]);
RND13(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[14]);
RND14(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[15]);
RND15(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[16]);
RND16(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[17]);
RND17(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[18]);
RND18(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[19]);
RND19(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[20]);
RND20(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[21]);
RND21(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[22]);
RND22(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[23]);
```

```
  RND23(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[24]);
  RND24(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[25]);
  RND25(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[26]);
  RND26(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[27]);
  RND27(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[28]);
  RND28(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[29]);
  RND29(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[30]);
  RND30(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[31]);
  RND31(x0, x1, x2, x3, y0, y1, y2, y3);
  x0 = y0; x1 = y1; x2 = y2; x3 = y3;
  keying(x0, x1, x2, x3, subkeys[32]);
  /* The ciphertext is now in x */

  ciphertext[0] = x0;
  ciphertext[1] = x1;
  ciphertext[2] = x2;
  ciphertext[3] = x3;
}

serpent_decrypt(unsigned long ciphertext[4],
                unsigned long plaintext[4],
                unsigned long subkeys[33][4])
{
  register unsigned long x0, x1, x2, x3;
  register unsigned long y0, y1, y2, y3;

  x0=ciphertext[0];
  x1=ciphertext[1];
  x2=ciphertext[2];
  x3=ciphertext[3];

  /* Start to decrypt the ciphertext x */
  keying(x0, x1, x2, x3, subkeys[32]);
  InvRND31(x0, x1, x2, x3, y0, y1, y2, y3);
  keying(y0, y1, y2, y3, subkeys[31]);
  inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
```

```
InvRND30(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[30]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND29(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[29]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND28(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[28]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND27(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[27]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND26(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[26]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND25(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[25]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND24(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[24]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND23(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[23]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND22(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[22]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND21(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[21]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND20(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[20]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND19(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[19]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND18(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[18]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND17(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[17]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND16(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[16]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND15(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[15]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND14(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[14]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND13(x0, x1, x2, x3, y0, y1, y2, y3);
```

```
  keying(y0, y1, y2, y3, subkeys[13]);
  inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
  InvRND12(x0, x1, x2, x3, y0, y1, y2, y3);
  keying(y0, y1, y2, y3, subkeys[12]);
  inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
  InvRND11(x0, x1, x2, x3, y0, y1, y2, y3);
  keying(y0, y1, y2, y3, subkeys[11]);
  inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
  InvRND10(x0, x1, x2, x3, y0, y1, y2, y3);
  keying(y0, y1, y2, y3, subkeys[10]);
  inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
  InvRND09(x0, x1, x2, x3, y0, y1, y2, y3);
  keying(y0, y1, y2, y3, subkeys[ 9]);
  inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
  InvRND08(x0, x1, x2, x3, y0, y1, y2, y3);
  keying(y0, y1, y2, y3, subkeys[ 8]);
  inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
  InvRND07(x0, x1, x2, x3, y0, y1, y2, y3);
  keying(y0, y1, y2, y3, subkeys[ 7]);
  inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
  InvRND06(x0, x1, x2, x3, y0, y1, y2, y3);
  keying(y0, y1, y2, y3, subkeys[ 6]);
  inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
  InvRND05(x0, x1, x2, x3, y0, y1, y2, y3);
  keying(y0, y1, y2, y3, subkeys[ 5]);
  inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
  InvRND04(x0, x1, x2, x3, y0, y1, y2, y3);
  keying(y0, y1, y2, y3, subkeys[ 4]);
  inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
  InvRND03(x0, x1, x2, x3, y0, y1, y2, y3);
  keying(y0, y1, y2, y3, subkeys[ 3]);
  inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
  InvRND02(x0, x1, x2, x3, y0, y1, y2, y3);
  keying(y0, y1, y2, y3, subkeys[ 2]);
  inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
  InvRND01(x0, x1, x2, x3, y0, y1, y2, y3);
  keying(y0, y1, y2, y3, subkeys[ 1]);
  inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
  InvRND00(x0, x1, x2, x3, y0, y1, y2, y3);
  x0 = y0; x1 = y1; x2 = y2; x3 = y3;
  keying(x0, x1, x2, x3, subkeys[ 0]);
  /* The plaintext is now in x */

  plaintext[0] = x0;
  plaintext[1] = x1;
  plaintext[2] = x2;
  plaintext[3] = x3;
}

#define min(x,y) (((x)<(y))?(x):(y))

int serpent_convert_from_string(int len, char *str, unsigned long *val)
```

```
/* the size of val must be at least the next multiple of 32 */
/* bits after len bits */
{
  int is, iv;
  int slen=min(strlen(str), (len+3)/4);

  if(len<0)
    return -1;           /* Error!!! */

  if(len>slen*4 || len<slen*4-3)
    return -1;           /* Error!!! */

  for(is=0; is<slen; is++)
    if(((str[is]<'0')||(str[is]>'9')) &&
       ((str[is]<'A')||(str[is]>'F')) &&
       ((str[is]<'a')||(str[is]>'f')))
      return -1; /* Error!!! */

  for(is=slen, iv=0; is>=8; is-=8, iv++)
    {
      unsigned long t;
      sscanf(&str[is-8], "%08lX", &t);
      val[iv] = t;
    }
  if(is>0)
    {
      char tmp[10];
      unsigned long t;
      strncpy(tmp, str, is);
      tmp[is] = 0;
      sscanf(tmp, "%08lX", &t);
      val[iv++] = t;
    }
  for(; iv<(len+31)/32; iv++)
    val[iv] = 0;
  return iv;
}

char *serpent_convert_to_string(int len, unsigned long val[8], char *str)
/* str must have at least (len+3)/4+1 bytes. */
{
  int i;

  if(len<0)
    return (char *)-1;           /* Error!!! */

  str[0] = 0;
  i=len/32;
  if(len&31>0)
    {
      char tmp[10];
      sprintf(tmp, "%08lX", val[i]&(((len&31)<<1)-1));
```

```
      strcat(str, &tmp[8-(((len&31)+3)/4)]);
    }
  for(i--; i>=0; i--)
    {
      char tmp[10];
      sprintf(tmp, "%08lX", val[i]);
      strcat(str, tmp);
    }
  return str;
}
```

Test.c

```
/*********************************************************/
/* Copyright (C) 1998 Ross Anderson, Eli Biham, Lars Knudsen
 * All rights reserved.
 *
 * This code is freely distributed for AES selection process.
 * No other use is allowed.
 *
 * Copyright remains of the copyright holders, and as such any Copyright
 * notices in the code are not to be removed.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted only for the AES selection process, provided
 * that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHORS OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 * The licence and distribution terms for any publically available version or
 * derivative of this code cannot be changed without the authors permission.
 *  i.e. this code cannot simply be copied and put under another distribution
 * licence [including the GNU Public Licence.]
 */
```

```
#define LOOPTIMES 1000000

#include "serpent.h"

char *serpent_convert_to_string();

main(int argc, char **argv)
{
  unsigned long x[4];
  int i, j;
  int loop_enc=LOOPTIMES, loop_dec=LOOPTIMES;
  char tmpstr[100];

  int rc;

  keyInstance keyI;
  cipherInstance cipherI;

  if(argc>1)
    loop_enc=atoi(argv[1]);
  if(argc>2)
    loop_dec=atoi(argv[2]);

  rc=cipherInit(&cipherI, MODE_ECB, "");
  if(rc<=0) exit(2);

  /* Let the plaintext be 0x00000000,0x00000001,0x00000002,0x00000003 */

  x[0] = 0; x[1] = 1; x[2] = 2; x[3] = 3;

  rc=makeKey(&keyI, DIR_ENCRYPT, 256,
           "0000000000000000000000000000000000000000000000000000000000000000");
  if(rc<=0) exit(2);

  printf("Plaintext          = %s¥n",
         serpent_convert_to_string(128, x, tmpstr));
  printf("Key                = %s¥n",
         serpent_convert_to_string(256, keyI.key, tmpstr));

  /* Repeatedly encrypt the plaintext LOOPTIMES times */
  for(i=0; i<loop_enc; i++)
    {
      rc=blockEncrypt(&cipherI, &keyI, (BYTE*)x, 128, (BYTE*)x);
      if(rc<=0) exit(2);
      if(i==0)
        printf("Ciphertext^%7d = %s¥n",
               i+1, serpent_convert_to_string(128, x, tmpstr));
    }

  printf("Ciphertext^%7d = %s¥n",
         loop_enc, serpent_convert_to_string(128, x, tmpstr));
```

```
  /* Repeatedly decrypt the plaintext LOOPTIMES times */
  for(i=loop_dec-1; i>=0; i--)
    {
      rc=blockDecrypt(&cipherI, &keyI, (BYTE*)x, 128, (BYTE*)x);
      if(rc<=0) exit(2);

      if(i==1)
        printf("Decrypted-Ciphertext=%s¥n",
               serpent_convert_to_string(128, x, tmpstr));
    }

  printf("Decrypted-plaintext= %s¥n",
         serpent_convert_to_string(128, x, tmpstr));

  exit(0);
}
```

2. SerpentEC のソースコード

serpensboxes.h は、前と同じものです。

Serpent.h

```
/*********************************************************/
/*  aes.h  */

/*  AES Cipher header file for ANSI C Submissions
      Lawrence E. Bassham III
      Computer Security Division
      National Institute of Standards and Technology

      April 15, 1998

    This sample is to assist implementers developing to the Cryptographic
API Profile for AES Candidate Algorithm Submissions.  Please consult this
document as a cross-reference.

    ANY CHANGES, WHERE APPROPRIATE, TO INFORMATION PROVIDED IN THIS FILE
MUST BE DOCUMENTED.  CHANGES ARE ONLY APPROPRIATE WHERE SPECIFIED WITH
THE STRING "CHANGE POSSIBLE".  FUNCTION CALLS AND THEIR PARAMETERS CANNOT
BE CHANGED.  STRUCTURES CAN BE ALTERED TO ALLOW IMPLEMENTERS TO INCLUDE
IMPLEMENTATION SPECIFIC INFORMATION.
*/
```

```
/*  Includes:
        Standard include files
*/

#include <stdio.h>

/*  Defines:
        Add any additional defines you need
*/

#define     DIR_ENCRYPT     0    /*  Are we encrpyting?  */
#define     DIR_DECRYPT     1    /*  Are we decrpyting?  */
#define     MODE_ECB        1    /*  Are we ciphering in ECB mode?   */
#define     MODE_CBC        2    /*  Are we ciphering in CBC mode?   */
#define     MODE_CFB1       3    /*  Are we ciphering in 1-bit CFB mode? */
#define     TRUE            1
#define     FALSE           0

/*  Error Codes - CHANGE POSSIBLE: inclusion of additional error codes  */
#define     BAD_KEY_DIR        -1  /*  Key direction is invalid, e;g;,
                                       unknown value */
#define     BAD_KEY_MAT        -2  /*  Key material not of correct
                                       length */
#define     BAD_KEY_INSTANCE   -3  /*  Key passed is not valid  */
#define     BAD_CIPHER_MODE    -4  /*  Params struct passed to
                                       cipherInit invalid */
#define     BAD_CIPHER_STATE   -5  /*  Cipher in wrong state (e.g., not
                                       initialized) */

/*  CHANGE POSSIBLE:  inclusion of algorithm specific defines  */
#define     MAX_KEY_SIZE 64  /* # of ASCII char's needed to
                                       represent a key */
#define     MAX_IV_SIZE          32  /* # of ASCII char's needed to
                                       represent an IV  */

/*  Typedefs:

        Typedef'ed data storage elements.  Add any algorithm specific
parameters at the bottom of the structs as appropriate.
*/

typedef    unsigned char    BYTE;

/*  The structure for key information */
typedef struct {
      BYTE  direction;    /*  Key used for encrypting or decrypting? */
      int   keyLen;       /*  Length of the key  */
      char  keyMaterial[MAX_KEY_SIZE+1];  /*  Raw key data in ASCII, e.g.,
                                       what the user types or KAT values)*/
      /*  The following parameters are algorithm dependent, replace or
                add as necessary  */
      unsigned long key[8];             /* The key in binary */
```

```
      unsigned long subkeys[33][4];       /* Serpent subkeys */
      } keyInstance;

/*  The structure for cipher information */
typedef struct {
      BYTE      mode;           /* MODE_ECB, MODE_CBC, or MODE_CFB1 */
      char  IV[MAX_IV_SIZE];     /* A possible Initialization Vector for
                                        ciphering */
      /*  Add any algorithm specific parameters needed here  */
      int   blockSize;           /* Sample: Handles non-128 bit block sizes
                                        (if available) */
      } cipherInstance;


/*  Function protoypes  */
int serpent_makeKey(keyInstance *key, BYTE direction, int keyLen,
            char *keyMaterial);

int cipherInit(cipherInstance *cipher, BYTE mode, char *IV);

int blockEncrypt(cipherInstance *cipher, keyInstance *key, BYTE *input,
                  int inputLen, BYTE *outBuffer);

int blockDecrypt(cipherInstance *cipher, keyInstance *key, BYTE *input,
                  int inputLen, BYTE *outBuffer);

int serpent_convert_from_string(int, char *, unsigned long *);
void serpent_encrypt(unsigned long [],unsigned long [], unsigned long [][4]);
void serpent_decrypt(unsigned long [],unsigned long [], unsigned long [][4]);
```

Serpent.cpp

```
/*********************************************************************/
/* Copyright (C) 1998 Ross Anderson, Eli Biham, Lars Knudsen
 * All rights reserved.
 *
 * This code is freely distributed for AES selection process.
 * No other use is allowed.
 *
 * Copyright remains of the copyright holders, and as such any Copyright
 * notices in the code are not to be removed.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted only for the AES selection process, provided
 * that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
```

```
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHORS OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 * The licence and distribution terms for any publically available version or
 * derivative of this code cannot be changed without the authors permission.
 *  i.e. this code cannot simply be copied and put under another distribution
 * licence [including the GNU Public Licence.]
 */

#include "stdafx.h"
#include "serpent.h"
#include "serpentsboxes.h"

#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

#define BLOCK_SIZE 128
#define DWORD unsigned long

/*  The functions  */
int serpent_makeKey(keyInstance *key, BYTE direction, int keyLen,
                          char *keyMaterial)
{
  unsigned long i,j;
  unsigned long w[132],k[132];
  int rc;

  if(direction != DIR_ENCRYPT &&
     direction != DIR_DECRYPT)
    return BAD_KEY_DIR;

  if(keyLen>256 || keyLen<1)
    return BAD_KEY_MAT;

  key->direction=direction;
  key->keyLen=keyLen;
  strncpy(key->keyMaterial, keyMaterial, MAX_KEY_SIZE+1);
```

```
rc=serpent_convert_from_string(keyLen, keyMaterial,key->key);
if(rc<=0)
  return BAD_KEY_MAT;

for(i=0; i<keyLen/32; i++)
  w[i]=key->key[i];
if(keyLen<256)
  w[i]=(key->key[i]&((1L<<((keyLen&31)))-1))|(1L<<((keyLen&31)));
for(i++; i<8; i++)
  w[i]=0;
for(i=8; i<16; i++)
  w[i]=ROL(w[i-8]^w[i-5]^w[i-3]^w[i-1]^PHI^(i-8),11);
for(i=0; i<8; i++)
  w[i]=w[i+8];
for(i=8; i<132; i++)
  w[i]=ROL(w[i-8]^w[i-5]^w[i-3]^w[i-1]^PHI^i,11);

RND03(w[  0], w[  1], w[  2], w[  3], k[  0], k[  1], k[  2], k[  3]);
RND02(w[  4], w[  5], w[  6], w[  7], k[  4], k[  5], k[  6], k[  7]);
RND01(w[  8], w[  9], w[ 10], w[ 11], k[  8], k[  9], k[ 10], k[ 11]);
RND00(w[ 12], w[ 13], w[ 14], w[ 15], k[ 12], k[ 13], k[ 14], k[ 15]);
RND31(w[ 16], w[ 17], w[ 18], w[ 19], k[ 16], k[ 17], k[ 18], k[ 19]);
RND30(w[ 20], w[ 21], w[ 22], w[ 23], k[ 20], k[ 21], k[ 22], k[ 23]);
RND29(w[ 24], w[ 25], w[ 26], w[ 27], k[ 24], k[ 25], k[ 26], k[ 27]);
RND28(w[ 28], w[ 29], w[ 30], w[ 31], k[ 28], k[ 29], k[ 30], k[ 31]);
RND27(w[ 32], w[ 33], w[ 34], w[ 35], k[ 32], k[ 33], k[ 34], k[ 35]);
RND26(w[ 36], w[ 37], w[ 38], w[ 39], k[ 36], k[ 37], k[ 38], k[ 39]);
RND25(w[ 40], w[ 41], w[ 42], w[ 43], k[ 40], k[ 41], k[ 42], k[ 43]);
RND24(w[ 44], w[ 45], w[ 46], w[ 47], k[ 44], k[ 45], k[ 46], k[ 47]);
RND23(w[ 48], w[ 49], w[ 50], w[ 51], k[ 48], k[ 49], k[ 50], k[ 51]);
RND22(w[ 52], w[ 53], w[ 54], w[ 55], k[ 52], k[ 53], k[ 54], k[ 55]);
RND21(w[ 56], w[ 57], w[ 58], w[ 59], k[ 56], k[ 57], k[ 58], k[ 59]);
RND20(w[ 60], w[ 61], w[ 62], w[ 63], k[ 60], k[ 61], k[ 62], k[ 63]);
RND19(w[ 64], w[ 65], w[ 66], w[ 67], k[ 64], k[ 65], k[ 66], k[ 67]);
RND18(w[ 68], w[ 69], w[ 70], w[ 71], k[ 68], k[ 69], k[ 70], k[ 71]);
RND17(w[ 72], w[ 73], w[ 74], w[ 75], k[ 72], k[ 73], k[ 74], k[ 75]);
RND16(w[ 76], w[ 77], w[ 78], w[ 79], k[ 76], k[ 77], k[ 78], k[ 79]);
RND15(w[ 80], w[ 81], w[ 82], w[ 83], k[ 80], k[ 81], k[ 82], k[ 83]);
RND14(w[ 84], w[ 85], w[ 86], w[ 87], k[ 84], k[ 85], k[ 86], k[ 87]);
RND13(w[ 88], w[ 89], w[ 90], w[ 91], k[ 88], k[ 89], k[ 90], k[ 91]);
RND12(w[ 92], w[ 93], w[ 94], w[ 95], k[ 92], k[ 93], k[ 94], k[ 95]);
RND11(w[ 96], w[ 97], w[ 98], w[ 99], k[ 96], k[ 97], k[ 98], k[ 99]);
RND10(w[100], w[101], w[102], w[103], k[100], k[101], k[102], k[103]);
RND09(w[104], w[105], w[106], w[107], k[104], k[105], k[106], k[107]);
RND08(w[108], w[109], w[110], w[111], k[108], k[109], k[110], k[111]);
RND07(w[112], w[113], w[114], w[115], k[112], k[113], k[114], k[115]);
RND06(w[116], w[117], w[118], w[119], k[116], k[117], k[118], k[119]);
RND05(w[120], w[121], w[122], w[123], k[120], k[121], k[122], k[123]);
RND04(w[124], w[125], w[126], w[127], k[124], k[125], k[126], k[127]);
RND03(w[128], w[129], w[130], w[131], k[128], k[129], k[130], k[131]);

for(i=0; i<=32; i++)
```

```
    for(j=0; j<4; j++)
      key->subkeys[i][j] = k[4*i+j];

  return TRUE;
}

int cipherInit(cipherInstance *cipher, BYTE mode, char *IV)
{
//  int i;
  int rc;

  if((mode != MODE_ECB) &&
     (mode != MODE_CBC) &&
     (mode != MODE_CFB1))
    return BAD_CIPHER_MODE;

  cipher->mode = mode;            /* MODE_ECB, MODE_CBC, or MODE_CFB1 */
  cipher->blockSize=128;
  if(mode != MODE_ECB)
    {
      rc=serpent_convert_from_string(cipher->blockSize, IV, (unsigned long*)cipher->IV);
      if(rc<=0)
        return BAD_CIPHER_STATE;
    }

  return TRUE;
}

int blockEncrypt(cipherInstance *cipher,
                 keyInstance *key,
                 BYTE *input,
                 int inputLen,
                 BYTE *outBuffer)
{
  unsigned long t[4];
  int  b, n, i;
  DWORD x[BLOCK_SIZE/32];
  BYTE bit, bit0, ctBit, carry;

  /*
   * Note about optimization: the code becomes slower of the calls to
   * serpent_encrypt and serpent_decrypt are replaced by inlined code.
   * (tested on Pentium 133MMX)
   */

  switch(cipher->mode)
    {
    case MODE_ECB:
      for(b=0; b<inputLen; b+=128, input+=16, outBuffer+=16)
        serpent_encrypt((unsigned long*)input,(unsigned long*) outBuffer, key->subkeys);
      return inputLen;
```

```
    case MODE_CBC:
      t[0] = ((unsigned long*)cipher->IV)[0];
      t[1] = ((unsigned long*)cipher->IV)[1];
      t[2] = ((unsigned long*)cipher->IV)[2];
      t[3] = ((unsigned long*)cipher->IV)[3];
      for(b=0; b<inputLen; b+=128, input+=16, outBuffer+=16)
        {
          t[0] ^= ((unsigned long*)input)[0];
          t[1] ^= ((unsigned long*)input)[1];
          t[2] ^= ((unsigned long*)input)[2];
          t[3] ^= ((unsigned long*)input)[3];
          serpent_encrypt(t, t, key->subkeys);
          ((unsigned long*)outBuffer)[0] = t[0];
          ((unsigned long*)outBuffer)[1] = t[1];
          ((unsigned long*)outBuffer)[2] = t[2];
          ((unsigned long*)outBuffer)[3] = t[3];
        }
      ((unsigned long*)cipher->IV)[0] = t[0];
      ((unsigned long*)cipher->IV)[1] = t[1];
      ((unsigned long*)cipher->IV)[2] = t[2];
      ((unsigned long*)cipher->IV)[3] = t[3];

      return inputLen;

    case MODE_CFB1:
                cipher->mode = MODE_ECB; /* do encryption in ECB */
                for (n=0;n<inputLen;n++)
                        {
                        blockEncrypt(cipher,key,(BYTE *)cipher->IV,BLOCK_SIZE,(BYTE *)x);
                        bit0  = 0x80 >> (n & 7);/* which bit position in byte */
                        ctBit = (input[n/8] & bit0) ^ ((((BYTE *) x)[0] & 0x80) >> (n&7));
                        outBuffer[n/8] = (outBuffer[n/8] & ~ bit0) | ctBit;
                        carry = ctBit >> (7 - (n&7));
                        for (i=BLOCK_SIZE/8-1;i>=0;i--)
                                {
                                bit = cipher->IV[i] >> 7;/* save next "carry" from shift */
                                cipher->IV[i] = (cipher->IV[i] << 1) ^ carry;
                                carry = bit;
                                }
                        }
                cipher->mode = MODE_CFB1;/* restore mode for next time */
                return inputLen;

    default:
      return BAD_CIPHER_STATE;
    }
}

int blockDecrypt(cipherInstance *cipher,
                 keyInstance *key,
                 BYTE *input,
                 int inputLen,
```

```
                BYTE *outBuffer)
{
  unsigned long t[4];
  int  b, n, i;
  DWORD x[BLOCK_SIZE/32];
  BYTE bit, bit0, ctBit, carry;

  switch(cipher->mode)
    {
    case MODE_ECB:
      for(b=0; b<inputLen; b+=128, input+=16, outBuffer+=16)
          serpent_decrypt( (unsigned long*)input, (unsigned long*)outBuffer, key->subkeys);
      return inputLen;

    case MODE_CBC:
      t[0] = ((unsigned long*)cipher->IV)[0];
      t[1] = ((unsigned long*)cipher->IV)[1];
      t[2] = ((unsigned long*)cipher->IV)[2];
      t[3] = ((unsigned long*)cipher->IV)[3];
      for(b=0; b<inputLen; b+=128, input+=16, outBuffer+=16)
        {
          serpent_decrypt((unsigned long*)input, (unsigned long*)outBuffer,key->subkeys);
          ((unsigned long*)outBuffer)[0] ^= t[0];
          ((unsigned long*)outBuffer)[1] ^= t[1];
          ((unsigned long*)outBuffer)[2] ^= t[2];
          ((unsigned long*)outBuffer)[3] ^= t[3];
          t[0] = ((unsigned long*)input)[0];
          t[1] = ((unsigned long*)input)[1];
          t[2] = ((unsigned long*)input)[2];
          t[3] = ((unsigned long*)input)[3];
        }
      ((unsigned long*)cipher->IV)[0] = t[0];
      ((unsigned long*)cipher->IV)[1] = t[1];
      ((unsigned long*)cipher->IV)[2] = t[2];
      ((unsigned long*)cipher->IV)[3] = t[3];
      return inputLen;

    case MODE_CFB1://blockDecrypt
                cipher->mode = MODE_ECB; /* do encryption in ECB */
                for (n=0;n<inputLen;n++)
                        {
                        blockEncrypt(cipher,key,(BYTE *)cipher->IV,BLOCK_SIZE,(BYTE *)x);
                        bit0  = 0x80 >> (n & 7);
                        ctBit = input[n/8] & bit0;
                        outBuffer[n/8] = (outBuffer[n/8] & ~ bit0) |
                                                        (ctBit ^ ((((BYTE *) x)[0] & 0x80) >> (n&7)));
                        carry = ctBit >> (7 - (n&7));
                        for (i=BLOCK_SIZE/8-1;i>=0;i--)
                                {
                                bit = cipher->IV[i] >> 7;/* save next "carry" from shift */
                                cipher->IV[i] = (cipher->IV[i] << 1) ^ carry;
                                carry = bit;
```

```
                                }
                        }
                cipher->mode = MODE_CFB1;/* restore mode for next time */
                return inputLen;

    default:
      return BAD_CIPHER_STATE;
    }
}

void serpent_encrypt(unsigned long plaintext[4],
                unsigned long ciphertext[4],
                unsigned long subkeys[33][4])
{
  register unsigned long x0, x1, x2, x3;
  register unsigned long y0, y1, y2, y3;

  x0=plaintext[0];
  x1=plaintext[1];
  x2=plaintext[2];
  x3=plaintext[3];

  /* Start to encrypt the plaintext x */
  keying(x0, x1, x2, x3, subkeys[ 0]);
  RND00(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[ 1]);
  RND01(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[ 2]);
  RND02(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[ 3]);
  RND03(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[ 4]);
  RND04(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[ 5]);
  RND05(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[ 6]);
  RND06(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[ 7]);
  RND07(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[ 8]);
  RND08(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[ 9]);
  RND09(x0, x1, x2, x3, y0, y1, y2, y3);
```

```
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[10]);
RND10(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[11]);
RND11(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[12]);
RND12(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[13]);
RND13(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[14]);
RND14(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[15]);
RND15(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[16]);
RND16(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[17]);
RND17(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[18]);
RND18(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[19]);
RND19(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[20]);
RND20(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[21]);
RND21(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[22]);
RND22(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[23]);
RND23(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[24]);
RND24(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[25]);
RND25(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[26]);
RND26(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
```

```
  keying(x0, x1, x2, x3, subkeys[27]);
  RND27(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[28]);
  RND28(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[29]);
  RND29(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[30]);
  RND30(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[31]);
  RND31(x0, x1, x2, x3, y0, y1, y2, y3);
  x0 = y0; x1 = y1; x2 = y2; x3 = y3;
  keying(x0, x1, x2, x3, subkeys[32]);
  /* The ciphertext is now in x */

  ciphertext[0] = x0;
  ciphertext[1] = x1;
  ciphertext[2] = x2;
  ciphertext[3] = x3;
}

void serpent_decrypt(unsigned long ciphertext[4],
                     unsigned long plaintext[4],
                     unsigned long subkeys[33][4])
{
  register unsigned long x0, x1, x2, x3;
  register unsigned long y0, y1, y2, y3;

  x0=ciphertext[0];
  x1=ciphertext[1];
  x2=ciphertext[2];
  x3=ciphertext[3];

  /* Start to decrypt the ciphertext x */
  keying(x0, x1, x2, x3, subkeys[32]);
  InvRND31(x0, x1, x2, x3, y0, y1, y2, y3);
  keying(y0, y1, y2, y3, subkeys[31]);
  inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
  InvRND30(x0, x1, x2, x3, y0, y1, y2, y3);
  keying(y0, y1, y2, y3, subkeys[30]);
  inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
  InvRND29(x0, x1, x2, x3, y0, y1, y2, y3);
  keying(y0, y1, y2, y3, subkeys[29]);
  inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
  InvRND28(x0, x1, x2, x3, y0, y1, y2, y3);
  keying(y0, y1, y2, y3, subkeys[28]);
  inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
  InvRND27(x0, x1, x2, x3, y0, y1, y2, y3);
  keying(y0, y1, y2, y3, subkeys[27]);
```

```
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND26(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[26]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND25(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[25]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND24(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[24]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND23(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[23]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND22(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[22]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND21(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[21]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND20(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[20]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND19(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[19]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND18(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[18]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND17(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[17]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND16(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[16]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND15(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[15]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND14(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[14]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND13(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[13]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND12(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[12]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND11(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[11]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND10(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[10]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
```

```
  InvRND09(x0, x1, x2, x3, y0, y1, y2, y3);
  keying(y0, y1, y2, y3, subkeys[ 9]);
  inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
  InvRND08(x0, x1, x2, x3, y0, y1, y2, y3);
  keying(y0, y1, y2, y3, subkeys[ 8]);
  inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
  InvRND07(x0, x1, x2, x3, y0, y1, y2, y3);
  keying(y0, y1, y2, y3, subkeys[ 7]);
  inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
  InvRND06(x0, x1, x2, x3, y0, y1, y2, y3);
  keying(y0, y1, y2, y3, subkeys[ 6]);
  inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
  InvRND05(x0, x1, x2, x3, y0, y1, y2, y3);
  keying(y0, y1, y2, y3, subkeys[ 5]);
  inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
  InvRND04(x0, x1, x2, x3, y0, y1, y2, y3);
  keying(y0, y1, y2, y3, subkeys[ 4]);
  inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
  InvRND03(x0, x1, x2, x3, y0, y1, y2, y3);
  keying(y0, y1, y2, y3, subkeys[ 3]);
  inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
  InvRND02(x0, x1, x2, x3, y0, y1, y2, y3);
  keying(y0, y1, y2, y3, subkeys[ 2]);
  inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
  InvRND01(x0, x1, x2, x3, y0, y1, y2, y3);
  keying(y0, y1, y2, y3, subkeys[ 1]);
  inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
  InvRND00(x0, x1, x2, x3, y0, y1, y2, y3);
  x0 = y0; x1 = y1; x2 = y2; x3 = y3;
  keying(x0, x1, x2, x3, subkeys[ 0]);
  /* The plaintext is now in x */

  plaintext[0] = x0;
  plaintext[1] = x1;
  plaintext[2] = x2;
  plaintext[3] = x3;
}

#define min(x,y) (((x)<(y))?(x):(y))

int serpent_convert_from_string(int len, char *str, unsigned long *val)
/* the size of val must be at least the next multiple of 32 */
/* bits after len bits */
{
  int is, iv;
  int slen=min((int)strlen(str), (len+3)/4);

  if(len<0)
    return -1;           /* Error!!! */

  if(len>slen*4 || len<slen*4-3)
    return -1;           /* Error!!! */
```

```
  for(is=0; is<slen; is++)
    if(((str[is]<'0')||(str[is]>'9')) &&
       ((str[is]<'A')||(str[is]>'F')) &&
       ((str[is]<'a')||(str[is]>'f')))
      return -1; /* Error!!! */

  for(is=slen, iv=0; is>=8; is-=8, iv++)
    {
      unsigned long t;
      sscanf(&str[is-8], "%08lX", &t);
      val[iv] = t;
    }
  if(is>0)
    {
      char tmp[10];
      unsigned long t;
      strncpy(tmp, str, is);
      tmp[is] = 0;
      sscanf(tmp, "%08lX", &t);
      val[iv++] = t;
    }
  for(; iv<(len+31)/32; iv++)
    val[iv] = 0;
  return iv;
}

char *serpent_convert_to_string(int len, unsigned long val[8], char *str)
/* str must have at least (len+3)/4+1 bytes. */
{
  int i;

  if(len<0)
    return (char *)-1;            /* Error!!! */

  str[0] = 0;
  i=len/32;
  if(len&31>0)
    {
      char tmp[10];
      sprintf(tmp, "%08lX", val[i]&(((len&31)<<1)-1));
      strcat(str, &tmp[8-(((len&31)+3)/4)]);
    }
  for(i--; i>=0; i--)
    {
      char tmp[10];
      sprintf(tmp, "%08lX", val[i]);
      strcat(str, tmp);
    }
  return str;
}
```

serpentEC.cpp

```
/*****************************************************************/
// SerpentEC.cpp : コンソールアプリケーション用のエントリポイントの定義
//

#include "stdafx.h"

#include <stdio.h>
#include <time.h>
#include <string.h>
#include <limits.h>
#include <stdlib.h>
#include <malloc.h>
#include "serpent.h"

#define file_len(x) (unsigned long)x

int s;
keyInstance keyI;
cipherInstance cipherI;


int main(int argc, char* argv[])
{
        FILE *fkey =0, *fin = 0, *fout = 0;
    fpos_t flen;
    unsigned long len, rlen, blen4;
        int mode,klen,blen,rc;
        char c_mode[4], c_klen[8], c_key[96], c_cini[64];
        unsigned char pbuf[2048], cbuf[2048];

        blen4 = 2048;

        printf( "Start!¥n" );

        if( argc != 4 ){                        // 使い方の誤り
                printf( "argc != 4 ¥n" );
                return -1;
        }

        if((fkey = fopen(argv[1], "rt")) == NULL){
                printf("Can not find key file for encryption. ¥n");
                return (-1);
        }

        fgets( c_mode, 4, fkey );
        fgets( c_klen, 8, fkey );
        fgets( c_key, 96, fkey );
```

```
	fgets( c_cini, 64, fkey );

	mode = atoi(c_mode);
	klen = atoi(c_klen);
	blen = 128;

	if(klen<56 || 256<klen){
		printf("Wrong key size. ¥n");
		return (-1);
	}

	/*Set mode*/
	if(mode == 1){
		rc=cipherInit(&cipherI, MODE_ECB, "");
	}
	if(mode == 2){
		rc=cipherInit(&cipherI, MODE_CBC, c_cini);
	}
	if(mode == 3){
		rc=cipherInit(&cipherI, MODE_CFB1, c_cini);
	}
	if(rc<=0){
		printf("モード設定が出来ません。");
		return(-2);
	}

	serpent_makeKey(&keyI, DIR_ENCRYPT,  klen, c_key );

	if((fin = fopen(argv[2], "rb")) == NULL){
		printf("Can not open plane text file. ¥n");
		return (-1);
	}

fseek(fin, 0, SEEK_END);
fgetpos(fin, &flen);
rlen = file_len(flen);
// reset to start
fseek(fin, 0, SEEK_SET);

	if((fout = fopen(argv[3], "wb")) == NULL){
		printf("Can not open encrypted data file. ¥n");
		return (-1);
	}

	s = sizeof(unsigned int);
	// write the bytes of the file
	*((unsigned int*)pbuf) = rlen;

	len = (unsigned long) fread(pbuf+s, 1, blen4-s , fin);
rlen -= len;
	rc=blockEncrypt(&cipherI, &keyI,(unsigned char*)(pbuf),8*blen4,(unsigned char*)(cbuf));
if(fwrite(cbuf, 1, blen4, fout) != (unsigned int)blen4)
```

```
        return -2;

    while(rlen > 0 && !feof(fin)){
        // read a block and reduce the remaining byte count
        len = (unsigned long)fread(pbuf, 1, blen4 , fin);
        rlen -= len;
                rc=blockEncrypt(&cipherI, &keyI,(unsigned char*)(pbuf),8*blen4,(unsigned
char*)(cbuf));
        if(fwrite(cbuf, 1, blen4, fout) != blen4){return -2;}
    }

        if(fkey){ fclose(fkey);  }
        if(fout){ fclose(fout);  }
        if(fin)  { fclose(fin);  }
        printf( "End!¥n" );
        return 0;
}
```

Stdafx.h

```
/*****************************************************************/
// stdafx.cpp : 標準インクルードファイルを含むソースファイル
//              SerpentEC.pch 生成されるプリコンパイル済ヘッダー
//              stdafx.obj 生成されるプリコンパイル済タイプ情報

#include "stdafx.h"

// TODO: STDAFX.H に含まれていて、このファイルに記述されていない
// ヘッダーファイルを追加してください。
```

以上、VC++2005 でソフトを作成するために必要なファイルです。

3. SerpentDC のソースコード

Serpentsboxes.h は前と同じものです。

Serpent.h

```
/*************************************************/
/*  aes.h  */

/*  AES Cipher header file for ANSI C Submissions
      Lawrence E. Bassham III
      Computer Security Division
```

```
      National Institute of Standards and Technology

      April 15, 1998

    This sample is to assist implementers developing to the Cryptographic
API Profile for AES Candidate Algorithm Submissions.  Please consult this
document as a cross-reference.

    ANY CHANGES, WHERE APPROPRIATE, TO INFORMATION PROVIDED IN THIS FILE
MUST BE DOCUMENTED.  CHANGES ARE ONLY APPROPRIATE WHERE SPECIFIED WITH
THE STRING "CHANGE POSSIBLE".  FUNCTION CALLS AND THEIR PARAMETERS CANNOT
BE CHANGED.  STRUCTURES CAN BE ALTERED TO ALLOW IMPLEMENTERS TO INCLUDE
IMPLEMENTATION SPECIFIC INFORMATION.
*/

/*  Includes:
        Standard include files
*/

#include <stdio.h>

/*  Defines:
        Add any additional defines you need
*/

#define     DIR_ENCRYPT     0    /*  Are we encrpyting?  */
#define     DIR_DECRYPT     1    /*  Are we decrpyting?  */
#define     MODE_ECB        1    /*  Are we ciphering in ECB mode?   */
#define     MODE_CBC        2    /*  Are we ciphering in CBC mode?   */
#define     MODE_CFB1       3    /*  Are we ciphering in 1-bit CFB mode? */
#define     TRUE            1
#define     FALSE           0

/*  Error Codes - CHANGE POSSIBLE: inclusion of additional error codes  */
#define     BAD_KEY_DIR        -1  /*  Key direction is invalid, e;g;,
                                       unknown value */
#define     BAD_KEY_MAT        -2  /*  Key material not of correct
                                       length */
#define     BAD_KEY_INSTANCE   -3  /*  Key passed is not valid  */
#define     BAD_CIPHER_MODE    -4  /*  Params struct passed to
                                       cipherInit invalid */
#define     BAD_CIPHER_STATE   -5  /*  Cipher in wrong state (e.g., not
                                       initialized) */

/*  CHANGE POSSIBLE:  inclusion of algorithm specific defines  */
#define     MAX_KEY_SIZE 64  /* # of ASCII char's needed to
                                       represent a key */
#define     MAX_IV_SIZE          32  /* # of ASCII char's needed to
                                       represent an IV  */

/*  Typedefs:
```

```
        Typedef'ed data storage elements.  Add any algorithm specific
parameters at the bottom of the structs as appropriate.
*/

typedef    unsigned char    BYTE;

/*  The structure for key information */
typedef struct {
      BYTE  direction;   /*  Key used for encrypting or decrypting? */
      int   keyLen;      /*  Length of the key  */
      char  keyMaterial[MAX_KEY_SIZE+1];  /*  Raw key data in ASCII, e.g.,
                                          what the user types or KAT values)*/
      /*  The following parameters are algorithm dependent, replace or
                add as necessary  */
      unsigned long key[8];             /* The key in binary */
      unsigned long subkeys[33][4];       /* Serpent subkeys */
      } keyInstance;

/*  The structure for cipher information */
typedef struct {
      BYTE       mode;           /* MODE_ECB, MODE_CBC, or MODE_CFB1 */
      char  IV[MAX_IV_SIZE];      /* A possible Initialization Vector for
                                          ciphering */
      /*  Add any algorithm specific parameters needed here  */
      int   blockSize;            /* Sample: Handles non-128 bit block sizes
                                          (if available) */
      } cipherInstance;


/*  Function protoypes  */
int serpent_makeKey(keyInstance *key, BYTE direction, int keyLen,
            char *keyMaterial);

int cipherInit(cipherInstance *cipher, BYTE mode, char *IV);

int blockEncrypt(cipherInstance *cipher, keyInstance *key, BYTE *input,
                 int inputLen, BYTE *outBuffer);

int blockDecrypt(cipherInstance *cipher, keyInstance *key, BYTE *input,
                 int inputLen, BYTE *outBuffer);

int serpent_convert_from_string(int, char *, unsigned long *);
void serpent_encrypt(unsigned long [],unsigned long [], unsigned long [][4]);
void serpent_decrypt(unsigned long [],unsigned long [], unsigned long [][4]);
```

Serpent.cpp

```
/*********************************************************************/
/* Copyright (C) 1998 Ross Anderson, Eli Biham, Lars Knudsen
```

```
 * All rights reserved.
 *
 * This code is freely distributed for AES selection process.
 * No other use is allowed.
 *
 * Copyright remains of the copyright holders, and as such any Copyright
 * notices in the code are not to be removed.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted only for the AES selection process, provided
 * that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHORS OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 * The licence and distribution terms for any publically available version or
 * derivative of this code cannot be changed without the authors permission.
 *  i.e. this code cannot simply be copied and put under another distribution
 * licence [including the GNU Public Licence.]
 */

#include "stdafx.h"
#include "serpent.h"
#include "serpentsboxes.h"

#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

#define BLOCK_SIZE 128
#define DWORD unsigned long

/*  The functions  */
int serpent_makeKey(keyInstance *key, BYTE direction, int keyLen,
                        char *keyMaterial)
```

```
{
  unsigned long i,j;
  unsigned long w[132],k[132];
  int rc;

  if(direction != DIR_ENCRYPT &&
     direction != DIR_DECRYPT)
    return BAD_KEY_DIR;

  if(keyLen>256 || keyLen<1)
    return BAD_KEY_MAT;

  key->direction=direction;
  key->keyLen=keyLen;
  strncpy(key->keyMaterial, keyMaterial, MAX_KEY_SIZE+1);

  rc=serpent_convert_from_string(keyLen, keyMaterial,key->key);
  if(rc<=0)
    return BAD_KEY_MAT;

  for(i=0; i<keyLen/32; i++)
    w[i]=key->key[i];
  if(keyLen<256)
    w[i]=(key->key[i]&((1L<<((keyLen&31)))-1))|(1L<<((keyLen&31)));
  for(i++; i<8; i++)
    w[i]=0;
  for(i=8; i<16; i++)
    w[i]=ROL(w[i-8]^w[i-5]^w[i-3]^w[i-1]^PHI^(i-8),11);
  for(i=0; i<8; i++)
    w[i]=w[i+8];
  for(i=8; i<132; i++)
    w[i]=ROL(w[i-8]^w[i-5]^w[i-3]^w[i-1]^PHI^i,11);

  RND03(w[  0], w[  1], w[  2], w[  3], k[  0], k[  1], k[  2], k[  3]);
  RND02(w[  4], w[  5], w[  6], w[  7], k[  4], k[  5], k[  6], k[  7]);
  RND01(w[  8], w[  9], w[ 10], w[ 11], k[  8], k[  9], k[ 10], k[ 11]);
  RND00(w[ 12], w[ 13], w[ 14], w[ 15], k[ 12], k[ 13], k[ 14], k[ 15]);
  RND31(w[ 16], w[ 17], w[ 18], w[ 19], k[ 16], k[ 17], k[ 18], k[ 19]);
  RND30(w[ 20], w[ 21], w[ 22], w[ 23], k[ 20], k[ 21], k[ 22], k[ 23]);
  RND29(w[ 24], w[ 25], w[ 26], w[ 27], k[ 24], k[ 25], k[ 26], k[ 27]);
  RND28(w[ 28], w[ 29], w[ 30], w[ 31], k[ 28], k[ 29], k[ 30], k[ 31]);
  RND27(w[ 32], w[ 33], w[ 34], w[ 35], k[ 32], k[ 33], k[ 34], k[ 35]);
  RND26(w[ 36], w[ 37], w[ 38], w[ 39], k[ 36], k[ 37], k[ 38], k[ 39]);
  RND25(w[ 40], w[ 41], w[ 42], w[ 43], k[ 40], k[ 41], k[ 42], k[ 43]);
  RND24(w[ 44], w[ 45], w[ 46], w[ 47], k[ 44], k[ 45], k[ 46], k[ 47]);
  RND23(w[ 48], w[ 49], w[ 50], w[ 51], k[ 48], k[ 49], k[ 50], k[ 51]);
  RND22(w[ 52], w[ 53], w[ 54], w[ 55], k[ 52], k[ 53], k[ 54], k[ 55]);
  RND21(w[ 56], w[ 57], w[ 58], w[ 59], k[ 56], k[ 57], k[ 58], k[ 59]);
  RND20(w[ 60], w[ 61], w[ 62], w[ 63], k[ 60], k[ 61], k[ 62], k[ 63]);
  RND19(w[ 64], w[ 65], w[ 66], w[ 67], k[ 64], k[ 65], k[ 66], k[ 67]);
  RND18(w[ 68], w[ 69], w[ 70], w[ 71], k[ 68], k[ 69], k[ 70], k[ 71]);
  RND17(w[ 72], w[ 73], w[ 74], w[ 75], k[ 72], k[ 73], k[ 74], k[ 75]);
```

```
  RND16(w[ 76], w[ 77], w[ 78], w[ 79], k[ 76], k[ 77], k[ 78], k[ 79]);
  RND15(w[ 80], w[ 81], w[ 82], w[ 83], k[ 80], k[ 81], k[ 82], k[ 83]);
  RND14(w[ 84], w[ 85], w[ 86], w[ 87], k[ 84], k[ 85], k[ 86], k[ 87]);
  RND13(w[ 88], w[ 89], w[ 90], w[ 91], k[ 88], k[ 89], k[ 90], k[ 91]);
  RND12(w[ 92], w[ 93], w[ 94], w[ 95], k[ 92], k[ 93], k[ 94], k[ 95]);
  RND11(w[ 96], w[ 97], w[ 98], w[ 99], k[ 96], k[ 97], k[ 98], k[ 99]);
  RND10(w[100], w[101], w[102], w[103], k[100], k[101], k[102], k[103]);
  RND09(w[104], w[105], w[106], w[107], k[104], k[105], k[106], k[107]);
  RND08(w[108], w[109], w[110], w[111], k[108], k[109], k[110], k[111]);
  RND07(w[112], w[113], w[114], w[115], k[112], k[113], k[114], k[115]);
  RND06(w[116], w[117], w[118], w[119], k[116], k[117], k[118], k[119]);
  RND05(w[120], w[121], w[122], w[123], k[120], k[121], k[122], k[123]);
  RND04(w[124], w[125], w[126], w[127], k[124], k[125], k[126], k[127]);
  RND03(w[128], w[129], w[130], w[131], k[128], k[129], k[130], k[131]);

  for(i=0; i<=32; i++)
    for(j=0; j<4; j++)
      key->subkeys[i][j] = k[4*i+j];

  return TRUE;
}

int cipherInit(cipherInstance *cipher, BYTE mode, char *IV)
{
//  int i;
  int rc;

  if((mode != MODE_ECB) &&
     (mode != MODE_CBC) &&
     (mode != MODE_CFB1))
    return BAD_CIPHER_MODE;

  cipher->mode = mode;            /* MODE_ECB, MODE_CBC, or MODE_CFB1 */
  cipher->blockSize=128;
  if(mode != MODE_ECB)
    {
      rc=serpent_convert_from_string(cipher->blockSize, IV, (unsigned long*)cipher->IV);
      if(rc<=0)
        return BAD_CIPHER_STATE;
    }

  return TRUE;
}

int blockEncrypt(cipherInstance *cipher,
                 keyInstance *key,
                 BYTE *input,
                 int inputLen,
                 BYTE *outBuffer)
{
  unsigned long t[4];
  int  b, n, i;
```

```
DWORD x[BLOCK_SIZE/32];
BYTE bit, bit0, ctBit, carry;

/*
 * Note about optimization: the code becomes slower of the calls to
 * serpent_encrypt and serpent_decrypt are replaced by inlined code.
 * (tested on Pentium 133MMX)
 */

switch(cipher->mode)
  {
  case MODE_ECB:
    for(b=0; b<inputLen; b+=128, input+=16, outBuffer+=16)
      serpent_encrypt((unsigned long*)input,(unsigned long*) outBuffer, key->subkeys);
    return inputLen;

  case MODE_CBC:
    t[0] = ((unsigned long*)cipher->IV)[0];
    t[1] = ((unsigned long*)cipher->IV)[1];
    t[2] = ((unsigned long*)cipher->IV)[2];
    t[3] = ((unsigned long*)cipher->IV)[3];
    for(b=0; b<inputLen; b+=128, input+=16, outBuffer+=16)
      {
        t[0] ^= ((unsigned long*)input)[0];
        t[1] ^= ((unsigned long*)input)[1];
        t[2] ^= ((unsigned long*)input)[2];
        t[3] ^= ((unsigned long*)input)[3];
        serpent_encrypt(t, t, key->subkeys);
        ((unsigned long*)outBuffer)[0] = t[0];
        ((unsigned long*)outBuffer)[1] = t[1];
        ((unsigned long*)outBuffer)[2] = t[2];
        ((unsigned long*)outBuffer)[3] = t[3];
      }
    ((unsigned long*)cipher->IV)[0] = t[0];
    ((unsigned long*)cipher->IV)[1] = t[1];
    ((unsigned long*)cipher->IV)[2] = t[2];
    ((unsigned long*)cipher->IV)[3] = t[3];

    return inputLen;

  case MODE_CFB1:
              cipher->mode = MODE_ECB; /* do encryption in ECB */
              for (n=0;n<inputLen;n++)
                      {
                      blockEncrypt(cipher,key,(BYTE *)cipher->IV,BLOCK_SIZE,(BYTE *)x);
                      bit0  = 0x80 >> (n & 7);/* which bit position in byte */
                      ctBit = (input[n/8] & bit0) ^ ((((BYTE *) x)[0] & 0x80) >> (n&7));
                      outBuffer[n/8] = (outBuffer[n/8] & ~ bit0) | ctBit;
                      carry = ctBit >> (7 - (n&7));
                      for (i=BLOCK_SIZE/8-1;i>=0;i--)
                              {
                              bit = cipher->IV[i] >> 7;/* save next "carry" from shift */
```

```
                                cipher->IV[i] = (cipher->IV[i] << 1) ^ carry;
                                carry = bit;
                                }
                        }
                cipher->mode = MODE_CFB1;/* restore mode for next time */
                return inputLen;

    default:
      return BAD_CIPHER_STATE;
    }
}

int blockDecrypt(cipherInstance *cipher,
                 keyInstance *key,
                 BYTE *input,
                 int inputLen,
                 BYTE *outBuffer)
{
  unsigned long t[4];
  int  b, n, i;
  DWORD x[BLOCK_SIZE/32];
  BYTE bit, bit0, ctBit, carry;

  switch(cipher->mode)
    {
    case MODE_ECB:
      for(b=0; b<inputLen; b+=128, input+=16, outBuffer+=16)
          serpent_decrypt( (unsigned long*)input, (unsigned long*)outBuffer, key->subkeys);
      return inputLen;

    case MODE_CBC:
      t[0] = ((unsigned long*)cipher->IV)[0];
      t[1] = ((unsigned long*)cipher->IV)[1];
      t[2] = ((unsigned long*)cipher->IV)[2];
      t[3] = ((unsigned long*)cipher->IV)[3];
      for(b=0; b<inputLen; b+=128, input+=16, outBuffer+=16)
        {
          serpent_decrypt((unsigned long*)input, (unsigned long*)outBuffer,key->subkeys);
          ((unsigned long*)outBuffer)[0] ^= t[0];
          ((unsigned long*)outBuffer)[1] ^= t[1];
          ((unsigned long*)outBuffer)[2] ^= t[2];
          ((unsigned long*)outBuffer)[3] ^= t[3];
          t[0] = ((unsigned long*)input)[0];
          t[1] = ((unsigned long*)input)[1];
          t[2] = ((unsigned long*)input)[2];
          t[3] = ((unsigned long*)input)[3];
        }
      ((unsigned long*)cipher->IV)[0] = t[0];
      ((unsigned long*)cipher->IV)[1] = t[1];
      ((unsigned long*)cipher->IV)[2] = t[2];
      ((unsigned long*)cipher->IV)[3] = t[3];
      return inputLen;
```

```
    case MODE_CFB1://blockDecrypt
                cipher->mode = MODE_ECB; /* do encryption in ECB */
                for (n=0;n<inputLen;n++)
                        {
                        blockEncrypt(cipher,key,(BYTE *)cipher->IV,BLOCK_SIZE,(BYTE *)x);
                        bit0  = 0x80 >> (n & 7);
                        ctBit = input[n/8] & bit0;
                        outBuffer[n/8] = (outBuffer[n/8] & ~ bit0) |
                                                        (ctBit ^ ((((BYTE *) x)[0] & 0x80) >> (n&7)));
                        carry = ctBit >> (7 - (n&7));
                        for (i=BLOCK_SIZE/8-1;i>=0;i--)
                                {
                                bit = cipher->IV[i] >> 7;/* save next "carry" from shift */
                                cipher->IV[i] = (cipher->IV[i] << 1) ^ carry;
                                carry = bit;
                                }
                        }
                cipher->mode = MODE_CFB1;/* restore mode for next time */
                return inputLen;

    default:
      return BAD_CIPHER_STATE;
    }
}

void serpent_encrypt(unsigned long plaintext[4],
                unsigned long ciphertext[4],
                unsigned long subkeys[33][4])
{
  register unsigned long x0, x1, x2, x3;
  register unsigned long y0, y1, y2, y3;

  x0=plaintext[0];
  x1=plaintext[1];
  x2=plaintext[2];
  x3=plaintext[3];

  /* Start to encrypt the plaintext x */
  keying(x0, x1, x2, x3, subkeys[ 0]);
  RND00(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[ 1]);
  RND01(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[ 2]);
  RND02(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[ 3]);
  RND03(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[ 4]);
```

```
RND04(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[ 5]);
RND05(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[ 6]);
RND06(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[ 7]);
RND07(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[ 8]);
RND08(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[ 9]);
RND09(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[10]);
RND10(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[11]);
RND11(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[12]);
RND12(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[13]);
RND13(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[14]);
RND14(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[15]);
RND15(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[16]);
RND16(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[17]);
RND17(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[18]);
RND18(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[19]);
RND19(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[20]);
RND20(x0, x1, x2, x3, y0, y1, y2, y3);
transform(y0, y1, y2, y3, x0, x1, x2, x3);
keying(x0, x1, x2, x3, subkeys[21]);
RND21(x0, x1, x2, x3, y0, y1, y2, y3);
```

```
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[22]);
  RND22(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[23]);
  RND23(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[24]);
  RND24(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[25]);
  RND25(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[26]);
  RND26(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[27]);
  RND27(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[28]);
  RND28(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[29]);
  RND29(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[30]);
  RND30(x0, x1, x2, x3, y0, y1, y2, y3);
  transform(y0, y1, y2, y3, x0, x1, x2, x3);
  keying(x0, x1, x2, x3, subkeys[31]);
  RND31(x0, x1, x2, x3, y0, y1, y2, y3);
  x0 = y0; x1 = y1; x2 = y2; x3 = y3;
  keying(x0, x1, x2, x3, subkeys[32]);
  /* The ciphertext is now in x */

  ciphertext[0] = x0;
  ciphertext[1] = x1;
  ciphertext[2] = x2;
  ciphertext[3] = x3;
}

void serpent_decrypt(unsigned long ciphertext[4],
                     unsigned long plaintext[4],
                     unsigned long subkeys[33][4])
{
  register unsigned long x0, x1, x2, x3;
  register unsigned long y0, y1, y2, y3;

  x0=ciphertext[0];
  x1=ciphertext[1];
  x2=ciphertext[2];
  x3=ciphertext[3];
```

```
/* Start to decrypt the ciphertext x */
keying(x0, x1, x2, x3, subkeys[32]);
InvRND31(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[31]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND30(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[30]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND29(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[29]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND28(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[28]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND27(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[27]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND26(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[26]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND25(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[25]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND24(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[24]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND23(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[23]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND22(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[22]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND21(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[21]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND20(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[20]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND19(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[19]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND18(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[18]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND17(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[17]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND16(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[16]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND15(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[15]);
```

```
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND14(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[14]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND13(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[13]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND12(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[12]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND11(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[11]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND10(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[10]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND09(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[ 9]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND08(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[ 8]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND07(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[ 7]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND06(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[ 6]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND05(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[ 5]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND04(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[ 4]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND03(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[ 3]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND02(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[ 2]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND01(x0, x1, x2, x3, y0, y1, y2, y3);
keying(y0, y1, y2, y3, subkeys[ 1]);
inv_transform(y0, y1, y2, y3, x0, x1, x2, x3);
InvRND00(x0, x1, x2, x3, y0, y1, y2, y3);
x0 = y0; x1 = y1; x2 = y2; x3 = y3;
keying(x0, x1, x2, x3, subkeys[ 0]);
/* The plaintext is now in x */

plaintext[0] = x0;
plaintext[1] = x1;
plaintext[2] = x2;
plaintext[3] = x3;
```

```
}

#define min(x,y) (((x)<(y))?(x):(y))

int serpent_convert_from_string(int len, char *str, unsigned long *val)
/* the size of val must be at least the next multiple of 32 */
/* bits after len bits */
{
  int is, iv;
  int slen=min((int)strlen(str), (len+3)/4);

  if(len<0)
    return -1;           /* Error!!! */

  if(len>slen*4 || len<slen*4-3)
    return -1;           /* Error!!! */

  for(is=0; is<slen; is++)
    if(((str[is]<'0')||(str[is]>'9')) &&
       ((str[is]<'A')||(str[is]>'F')) &&
       ((str[is]<'a')||(str[is]>'f')))
      return -1; /* Error!!! */

  for(is=slen, iv=0; is>=8; is-=8, iv++)
    {
      unsigned long t;
      sscanf(&str[is-8], "%08lX", &t);
      val[iv] = t;
    }
  if(is>0)
    {
      char tmp[10];
      unsigned long t;
      strncpy(tmp, str, is);
      tmp[is] = 0;
      sscanf(tmp, "%08lX", &t);
      val[iv++] = t;
    }
  for(; iv<(len+31)/32; iv++)
    val[iv] = 0;
  return iv;
}

char *serpent_convert_to_string(int len, unsigned long val[8], char *str)
/* str must have at least (len+3)/4+1 bytes. */
{
  int i;

  if(len<0)
    return (char *)-1;           /* Error!!! */

  str[0] = 0;
```

```
  i=len/32;
  if(len&31>0)
    {
      char tmp[10];
      sprintf(tmp, "%08lX", val[i]&(((len&31)<<1)-1));
      strcat(str, &tmp[8-(((len&31)+3)/4)]);
    }
  for(i--; i>=0; i--)
    {
      char tmp[10];
      sprintf(tmp, "%08lX", val[i]);
      strcat(str, tmp);
    }
  return str;
}
```

SerpentDC.cpp

```
/*****************************************************************************/
// SerpentDC.cpp : コンソールアプリケーション用のエントリポイントの定義
//

#include "stdafx.h"

#include <stdio.h>
#include <time.h>
#include <string.h>
#include <limits.h>
#include <stdlib.h>
#include <malloc.h>
#include "serpent.h"

#define file_len(x) (unsigned long)x

int s;
keyInstance keyI;
cipherInstance cipherI;

int main(int argc, char* argv[])
{
        FILE *fkey , *fin , *fout ;
    fpos_t flen;
    unsigned long len, rlen, blen4, pfilelen;
        int mode,klen,blen,rc;
        char c_mode[4], c_klen[8], c_key[96], c_cini[64];
        unsigned char cbuf[2048], pbuf[2048];

        blen4 = 2048;
```

```
printf( "Start!¥n" );

if( argc != 4 ){                          // 使い方の誤り
        printf( "argc != 4 ¥n" );
        return -1;
}

if((fkey = fopen(argv[1], "rt")) == NULL){
        printf("Can not find key file for decryption. ¥n");
        return (-1);
}

fgets( c_mode, 4, fkey );
fgets( c_klen, 8, fkey );
fgets( c_key, 96, fkey );
fgets( c_cini, 64, fkey );

mode = atoi(c_mode);
klen = atoi(c_klen);
blen = 128;

if(klen<56 || 256<klen){
        printf("Wrong key size. ¥n");
        return (-1);
}

/*Set mode*/
if(mode == 1){
        rc=cipherInit(&cipherI, MODE_ECB, "");
}
if(mode == 2){
        rc=cipherInit(&cipherI, MODE_CBC, c_cini);
}
if(mode == 3){
        rc=cipherInit(&cipherI, MODE_CFB1, c_cini);
}
if(rc<=0){
        printf("モード設定が出来ません。");
        return(-2);
}
serpent_makeKey(&keyI, DIR_DECRYPT,  klen, c_key );


if((fin = fopen(argv[2], "rb")) == NULL){
        printf("Can not open crypted file. ¥n");
        return (-1);
}
if((fout = fopen(argv[3], "wb")) == NULL){
        printf("Can not open plane data file. ¥n");
        return (-1);
}
```

```
    fseek(fin, 0, SEEK_END);
    fgetpos(fin, &flen);
    rlen = file_len(flen);
    // reset to start
    fseek(fin, 0, SEEK_SET);
        s = sizeof(unsigned long);
        // write the bytes of the file
        len = (unsigned long) fread(cbuf, 1, blen4 , fin);
        rlen = rlen - len;

        if(len < blen4){ return -3;       }

        rc=blockDecrypt(&cipherI, &keyI,(unsigned char*)(cbuf),8*blen4,(unsigned char*)(pbuf));
                // 復号文出力
        pfilelen = *((long*)(pbuf));

        if(pfilelen <= blen4 - s){
                if(fwrite(pbuf+s, 1, pfilelen, fout) != pfilelen){return -2;}
                if(fkey){ fclose(fkey);  }
                if(fout){ fclose(fout);  }
                if(fin)  { fclose(fin);  }
                printf( "End!¥n" );
                return 0;
        }
        else{
                if(fwrite(pbuf+s, 1, blen4 - s, fout) != blen4 - s){return -2;}
                pfilelen -= (blen4 - s);
        }

    if((rlen <= blen4) && (rlen > 0))
        {       // if the file length is less than or equal to 2048 bytes
        len = (unsigned long) fread(cbuf, 1, blen4 , fin);
        rlen -= len;
        if(rlen > 0){ return -2; }
                rc=blockDecrypt(&cipherI, &keyI,(unsigned char*)(cbuf),8*blen4,(unsigned
char*)(pbuf));
                if(fwrite(pbuf, 1, pfilelen, fout) != pfilelen){ return -2; }
                if(fkey){ fclose(fkey);  }
                if(fout){ fclose(fout);  }
                if(fin)  { fclose(fin);  }
                printf( "End!¥n" );
                return 0;
    }
    else
    {   // if the file length is more 1024 bytes
        // read the file a block at a time
        while(rlen > 0 && !feof(fin))
        {
            // read a block and reduce the remaining byte count
            len = (unsigned long)fread(cbuf, 1, blen4 , fin);
            rlen -= len;
            if((rlen>0) && (len==blen4)){
```

```
                                rc=blockDecrypt(&cipherI, &keyI,(unsigned
char*)(cbuf),8*blen4,(unsigned char*)(pbuf));
                                if(fwrite(pbuf, 1, blen4, fout) != blen4){return -2;}
                                pfilelen -= blen4;
                            }
                if(rlen<=0){
                                rc=blockDecrypt(&cipherI, &keyI,(unsigned
char*)(cbuf),8*blen4,(unsigned char*)(pbuf));
                                if(fwrite(pbuf, 1, pfilelen, fout) != pfilelen){return -2;}
                                if(fkey){ fclose(fkey);  }
                                if(fout){ fclose(fout);  }
                                if(fin)  { fclose(fin);  }
                                printf( "End!¥n" );
                                return 0;
                            }
            }
        }
         if(fkey){ fclose(fkey);  }
         if(fout){ fclose(fout);  }
         if(fin)  { fclose(fin);  }
         printf( "End!¥n" );
         return 0;
}
```

Stdafx.cpp

```
/**************************************************************************/
// stdafx.cpp : 標準インクルードファイルを含むソースファイル
//              SerpentDC.pch 生成されるプリコンパイル済ヘッダー
//              stdafx.obj 生成されるプリコンパイル済タイプ情報

#include "stdafx.h"

// TODO: STDAFX.H に含まれていて、このファイルに記述されていない
// ヘッダーファイルを追加してください。
```

Stdafx.h

```
/**************************************************************/
// stdafx.h : 標準のシステムインクルードファイル、
//            または参照回数が多く、かつあまり変更されない
//            プロジェクト専用のインクルードファイルを記述します。
//

#if !defined(AFX_STDAFX_H__08F0068B_7015_4F63_A50A_35A9784E5640__INCLUDED_)
#define AFX_STDAFX_H__08F0068B_7015_4F63_A50A_35A9784E5640__INCLUDED_

#if _MSC_VER > 1000
```

```
#pragma once
#endif // _MSC_VER > 1000

#define WIN32_LEAN_AND_MEAN // Windows ヘッダーから殆ど使用されないスタッフを除外します

#include <stdio.h>

// TODO: プログラムで必要なヘッダー参照を追加してください。

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ は前行の直前に追加の宣言を挿入します。

#endif // !defined(AFX_STDAFX_H__08F0068B_7015_4F63_A50A_35A9784E5640__INCLUDED_)
```

おわり。　　2012.04.08　　宇山靖政